Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки

«Затверджую»
в.о. завідуючого кафедри
комп'ютерних систем та робототехніки
_____ к. ф.-м. н., доцент Максим Хруслов
«___» червня 2025 р.

Пояснювальна записка

до кваліфікаційної роботи
бакалавра

на тему: «Backend-система управління сценаріями та виборами користувачів у графічних пригодницьких іграх»

Спеціальність 151 –Автоматизація та комп'ютерно-інтегровані технології
Галузь знань 15 – Автоматизація та приладобудування
Освітня програма «Автоматизація та комп'ютерно-інтегровані технології»

Захищено на засіданні
Екзаменаційної комісії № 46
протокол № __ від __.06.2025 р.
Оцінка _____ / _____
Голова Екзаменаційної комісії
_____ **ЧУГАЙ А.М.**

Виконав:
Студент групи КУ– 41
САВЧУК Владислав Артемович _____

Керівник: к.т.н., доцент, доцент ЗВО кафедри комп'ютерних систем та робототехніки
БИКОВА Тетяна Володимирівна _____

Рецензент: к.т.н., доцент, доцент ЗВО кафедри мат. моделювання та аналізу даних
КОРОБЧИНСЬКИЙ Кирил Петрович _____

Харків – 2025

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи бакалавра складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і чотирьох додатків. Загальний обсяг роботи складає 80 сторінки, із яких 48 сторінки основної частини з 20 рисунками, 1 таблицею, 17 найменуваннями списку використаних джерел та чотирма додатками.

Метою кваліфікаційної роботи є вирішення проблеми складності та доступності реалізації інтерактивних сценаріїв у комп'ютерних іграх через впровадження модульної системи, орієнтованої на потреби інді-розробників.

Об'єкт дослідження – процес проектування та реалізації системи для управління сценаріями та виборами гравців у відеоіграх.

Предмет дослідження – архітектура та функціонал backend системи для управління сценаріями, які забезпечують інтерактивність, нелінійність та врахування вибору користувача в графічних пригодницьких іграх.

Проблема, яку вирішує кваліфікаційна робота, полягає відсутності доступного, гнучкого та масштабованого інструменту для інтеграції механізмів управління сценаріями в іграх. Існуючі рішення часто є складними у використанні та підтримці, мають обмежений функціонал або є платними, що створює бар'єри для інді-розробників.

Область застосування – розробка комп'ютерних ігор з інтерактивними сюжетами. Backend система може бути використана інді-розробниками, ігровими студіями та платформами для створення графічних пригодницьких ігор з нелінійними сценаріями.

Ключові слова: backend система, Unity, C#, графічна пригодницька гра, інтерактивний сюжет, сценарії, вибір користувача, програмування.

ANNOTATION

The explanatory note for the bachelor's qualification work consists of an introduction, four chapters, conclusions, a list of references, and four appendices. The total volume of the work is 80 pages, including 48 pages of the main part with 20 drawings, 1 tables, 17 references, and four appendices.

The goal of the qualification work is to solve the problem of the complexity and accessibility of implementing interactive scenarios in computer games through the implementation of a modular system focused on the needs of indie developers.

The object of research is the process of designing and implementing a system for a managing scenarios and player choices in video games.

The subject of research is the architecture and functionality of the backend system for managing scenarios that ensure interactivity, non-linearity, and consideration of user choices in graphic adventure games.

The problem that the qualification work addresses is the lack of an accessible, flexible, and scalable tool for integrating script management mechanisms into games. Existing solutions are often difficult to use and maintain, have limited functionality, or are paid, which creates barriers for indie developers.

The area of application is the development of computer games with interactive plots. The backend system can be utilized by indie developers, game studios, and platforms for creating graphic adventure games with non-linear scenarios.

***Keywords*:** backend system, Unity, C#, graphic adventure game, interactive plot, scenarios, user choice, programming.

ЗМІСТ

ВСТУП

У сучасному світі відеоігри стали невід'ємною частиною розважальної індустрії, що активно розвивається. Зростаючий попит на інтерактивні сюжети, які дозволяють користувачам впливати на розвиток подій у грі, вимагає створення ефективних рішень для управління сценаріями та виборами гравців. Особливу увагу привертають графічні пригодницькі ігри, де головним елементом є залучення користувача через унікальні історії, які реагують на їх вибори.

Актуальність роботи. Створення backend системи для інтерактивного управління сценаріями у відеоіграх дозволяє забезпечити нелінійний розвиток сюжету, збільшуючи глибину взаємодії гравця з ігровим світом. Незважаючи на наявність існуючих рішень, багато з них мають обмеження щодо масштабованості, зручності інтеграції та адаптації під конкретні потреби розробників. Використання сучасних технологій, таких як рушій Unity та мова програмування C#, відкриває можливості для створення оптимізованої, модульної та адаптивної системи управління сценаріями.

Метою дослідження є вирішення проблеми складності та доступності реалізації інтерактивних сценаріїв у комп’ютерних іграх через впровадження модульної системи, орієнтованої на потреби інді-розробників.

Об’єкт дослідження – процес проектування та реалізації системи для управління сценаріями та виборами гравців у відеоіграх.

Методи дослідження: методи аналізу існуючих рішень, розробка архітектури системи, тестування інтерактивних механік.

Предмет дослідження – архітектура та функціональні можливості backend системи для забезпечення інтерактивності, нелінійності та адаптивності ігрових сценаріїв.

Завдання дослідження:

1. Проаналізувати існуючі рішення для управління сценаріями в графічних пригодницьких іграх.
2. Розробити архітектуру backend системи, яка базується на сучасних підходах до побудови програмного забезпечення.
3. Реалізувати функціонал для роботи з інтерактивними діалогами, виборами користувачів та їх наслідками.
4. Інтегрувати систему залежностей з використанням Zenject для забезпечення гнучкості та масштабованості проекту.
5. Провести тестування backend системи на базі створеної гри та оцінити її ефективність.

РОЗДІЛ 1
ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1. Аналіз сучасних backend систем для інтерактивних ігор

Backend системи для інтерактивних ігор забезпечують основу для реалізації складних сюжетів, які змінюються залежно від виборів гравця. Вони дозволяють створювати інтерактивні, нелінійні історії, які підвищують рівень занурення користувачів у ігровий процес. У графічних пригодницьких іграх ця функціональність є критично важливою, оскільки основний акцент робиться на сюжеті та виборі.

На сучасному ринку можна знайти декілька підходів і рішень для побудови таких систем:

Сучасні аналоги backend систем

Діалогові системи на основі JSON або XML

Переваги:

- Простота у використанні та внесенні змін: файли JSON або XML легко редагуються навіть без глибоких технічних знань.

- Легка інтеграція з багатьма рушіями, включаючи Unity.

- Можливість створення великих розгалужених сценаріїв із мінімальними витратами ресурсів.

Недоліки:

- Відсутність вбудованих механізмів для забезпечення логіки сценаріїв (перевірки станів, залежностей).

- Обмежені можливості для масштабування у великих проектах, що вимагають інтеграції складних залежностей.

Квестові системи в іграх "The Walking Dead", "Life is Strange", "Detroit: Become Human"

Переваги:

- Забезпечують високий рівень інтерактивності.

- Використовують унікальні системи для відстеження виборів гравця та наслідків цих виборів.

- Підтримка нелінійного сюжету з різними кінцівками.

Недоліки:

- Реалізація таких систем часто є внутрішньою і недоступна для інших розробників.

- Високий рівень складності підтримки великих сценаріїв.

- Велика вартість розробки через складність інтеграції багатьох механік.

NodeCanvas і Dialogue System for Unity

NodeCanvas – це інструмент для побудови логіки на основі графів.

Переваги:

- Інтуїтивний графічний інтерфейс для побудови сценаріїв.

- Підтримка нелінійних сценаріїв.

Недоліки:

- Обмежена гнучкість, якщо необхідна складна логіка.

- Необхідність додаткового налаштування для інтеграції у специфічні проекти.

Dialogue System for Unity – популярна система для створення діалогів.

Переваги:

- Можливість створення складних діалогів із розгалуженнями.

- Інтеграція з Unity та інші корисні функції.

Недоліки:

- Складність масштабування у великих проектах.

- Висока залежність від інструментів Unity, що ускладнює використання в інших середовищах.

Аналоги на Unity Asset Store

На Unity Asset Store представлено небагато високоякісних систем управління сценаріями. Більшість доступних рішень є або платними, або

застарілими, які використовують методи, на які Unity викидає попередження. Серед безкоштовних рішень є лише одна система, яка забезпечує базовий функціонал і підходить переважно для простих ігор з нескладною логікою, до того ж ця система була написана під версію 2018.2.12f1, що являється досить застарілою.

Переваги існуючих рішень:

- Готові до використання без необхідності розробки з нуля.

- Деякі з них мають вбудовані редактори сценаріїв із графічним інтерфейсом.

- Інтегровані з іншими інструментами Unity, такими як анімація та UI.

Недоліки:

- Висока вартість багатьох сучасних рішень.

- Відсутність широких можливостей для складних та нелінійних сценаріїв у безкоштовних версіях.

- Розроблені під старі версії Unity безкоштові та навіть деякі платні системи.

- Обмежені можливості кастомізації, що ускладнює адаптацію під конкретні потреби проектів.

Недоліки сучасних рішень

- **Масштабованість.** У великих проектах часто виникають труднощі з підтримкою великих сценаріїв. Наприклад, розгалужені діалоги та складна логіка можуть зробити систему громіздкою, що ускладнює внесення змін.

- **Гнучкість.** Багато рішень, особливо комерційних, пропонують обмежений набір функцій, що змушує розробників адаптувати проект до можливостей системи, а не навпаки.

- **Інтеграція.** Існуючі системи можуть вимагати значного часу на налаштування та адаптацію до специфіки проекту.

Вимоги до системи, яку ми розробляємо

На основі аналізу сучасних рішень можна сформулювати основні вимоги до backend системи, яка створюється спеціально для Unity:

- **Гнучкість у побудові сценаріїв.** Система має підтримувати складні розгалуження сюжету без зайвих обмежень.

- **Простота використання.** Навіть у складних проектах розробники повинні мати можливість швидко змінювати або додавати сценарії.

- **Інтеграція з Unity.** Повна сумісність із рушієм, включаючи використання вбудованих можливостей, таких як анімації, Canvas і Zenject для управління залежностями.

- **Оптимізація.** Система повинна бути легкою для роботи навіть на пристроях із середніми характеристиками.

Розробка власної системи, яка відповідає цим вимогам, дозволить усунути обмеження існуючих рішень, забезпечуючи високу адаптивність, інтерактивність та легкість інтеграції у майбутні проекти.

1.2. Порівняння Unity та інших рушіїв (Unreal Engine, Godot, CryEngine)

Вибір рушія для реалізації нашого проєкту є критично важливим, адже від нього залежить не лише ефективність розробки, але й можливість реалізувати всі поставлені задачі: нелінійність сюжету, інтерактивність виборів користувачів і оптимізацію для різних платформ. Серед популярних рушіїв, які активно використовуються у галузі, виділяються Unity, Unreal Engine, Godot і CryEngine. Кожен із них має свої переваги та недоліки, які визначають їхню придатність для реалізації конкретних задач.

Unity

Unity – це один із найбільш універсальних рушіїв, який використовується як інді-розробниками, так і великими студіями.

Переваги:

- **Мультиплатформність:** Підтримує понад 25 платформ, включаючи ПК, мобільні пристрої, консолі, VR/AR тощо. Це дає можливість легко адаптувати наш проєкт для різних пристроїв.

- **Гнучкість:** Завдяки URP та HDRP Unity дозволяє налаштовувати графіку відповідно до потреб: від базової оптимізації для слабших пристроїв до високоякісної графіки.

- **Зручність програмування:** Використання C# як основної мови дозволяє швидко реалізовувати функціонал та забезпечує високу читабельність коду.

- **Розвинена екосистема:** Asset Store пропонує величезну кількість готових рішень, які можуть бути адаптовані до нашого проєкту.

- **Документація та спільнота:** Unity має одну з найбільших у світі розробницьких спільнот, що забезпечує швидкий доступ до інформації та вирішення проблем.

Недоліки:

- **Обмежена графіка:** Попри суттєвий прогрес, Unity поступається Unreal Engine у створенні фотореалістичних сцен.

- **Проблеми з великими відкритими світами:** Unity більше підходить для ігор із помірним рівнем деталізації та меншою кількістю об’єктів.

Unreal Engine

Unreal Engine є одним із провідних рушіїв для створення ігор із реалістичною графікою та складними механіками.

Переваги:

- **Графіка:** Unreal Engine забезпечує фотореалістичну графіку завдяки передовим технологіям, таким як Lumen та Nanite.

- **Масштабність:** Підходить для створення великих відкритих світів із високим рівнем деталізації.

- **Blueprints:** Інструмент для візуального програмування, який дозволяє реалізувати складну логіку без глибоких знань програмування.

Недоліки:

- **Складність освоєння:** Використання C++ та специфічних налаштувань вимагає значного досвіду та часу.

- **Високі системні вимоги:** Unreal Engine вимагає потужного обладнання як для розробників, так і для кінцевих користувачів.

Godot

Godot є рушієм із відкритим вихідним кодом, орієнтованим на 2D- та невеликі 3D-проєкти.

Переваги:

- **Повна безкоштовність:** Відсутність ліцензійних обмежень.

- **Легкість освоєння:** Інтерфейс і документація адаптовані для новачків.

- **Низькі системні вимоги:** Godot працює навіть на старих пристроях.

Недоліки:

- **Обмеження для складних 3D-проєктів:** Підходить переважно для простих ігор.

- **Мала спільнота:** Обмежена кількість готових рішень і навчальних матеріалів.

CryEngine

CryEngine спеціалізується на високоякісній графіці та великих відкритих світах.

Переваги:

- **Графіка:** Високий рівень деталізації завдяки передовим алгоритмам освітлення.

- **Підтримка відкритих світів:** Ідеально підходить для створення масштабних ігрових просторів.

Недоліки:

- **Складність освоєння:** Вимагає знання C++ та спеціалізованих інструментів.

- **Менша популярність:** Відсутність широкої спільноти розробників та обмежена документація.

Таблиця 1.1.

Порівняння рушіїв

Характеристика	Unity	Unreal Engine	Godot	CryEngine
Мова програмування	C#	C++, Blueprints	GDScript, C#	C++
Простота освоєння	Висока	Низька	Дуже висока	Низька
Графіка	Хороша	Фотореалістична	Середня	Фотореалістична
Мульти-платформність	25+ платформ	10+ платформ	10+ платформ	5+ платформ
Вартість	Безкоштовно (умови)	Безкоштовно (умови)	Повністю безкоштовно	Безкоштовно (умови)
Екосистема	Велика	Велика	Мала	Середня
Оптимізація	Гнучка	Вимоглива	Помірна	Вимоглива

Чому обрано саме Unity

Unity є ідеальним вибором для нашого проєкту з таких причин:

- **Простота реалізації.** Завдяки використанню C# розробка логіки виборів і сценаріїв стає швидшою, інтуїтивнішою та менш затратною за часом порівняно з мовами програмування інших рушіїв, таких як C++ в Unreal Engine чи CryEngine.

- **Оптимізація.** Unity дозволяє адаптувати проєкт до пристроїв із середньою продуктивністю, забезпечуючи плавну роботу навіть на комп'ютерах з обмеженими ресурсами.
- **Розвинена спільнота.** Величезна база знань, велика кількість навчальних матеріалів, готових рішень у Asset Store та активна спільнота розробників забезпечують швидке вирішення проблем, які можуть виникнути під час розробки.

Таким чином, Unity ідеально відповідає потребам нашого проєкту, забезпечуючи всі необхідні інструменти для створення інтерактивного нелінійного сюжету, оптимізацію для різних пристроїв та підтримку стилізованої графіки. Цей рушій поєднує гнучкість, зручність та продуктивність, що робить його найкращим вибором для реалізації завдань.

1.3. Проблеми та обмеження існуючих систем

Сучасні backend системи для інтерактивних ігор відіграють ключову роль у забезпеченні нелінійності сюжету та управлінні виборами користувачів. Проте навіть найпопулярніші рішення стикаються з низкою проблем і обмежень, які впливають на їхню ефективність, гнучкість і зручність інтеграції.

Проблеми з масштабуванням сценаріїв

Ігри з багатогалузевими сценаріями стикаються зі складнощами в управлінні та розширенні сценаріїв.

Ускладнення структури сценаріїв:

- Backend системи, що базуються на JSON або XML, можуть стати надто складними для обробки, коли кількість виборів і розгалужень суттєво зростає.

- Відсутність інструментів для автоматизованого зв’язування подій і перевірки логіки призводить до помилок у сценаріях.

Висока трудомісткість: Підтримка великих сценаріїв стає складною через необхідність постійного оновлення структури файлів вручну.

Обмеження у роботі з виборами користувачів

Backend системи мають забезпечувати динамічну реакцію на вибори гравців, але часто стикаються з такими проблемами:

Нестача адаптивності:

- Деякі системи не враховують складні взаємозалежності між виборами гравців, що обмежує нелінійність сюжету.

- Відсутність механізмів для відстеження глобальних змін, які впливають на всі аспекти гри, а не лише на локальні події.

Фіксованість архітектури: Системи з жорстко заданими правилами часто обмежують можливості розширення функціоналу для нових механік чи сценаріїв.

Складність інтеграції

Існуючі рішення не завжди забезпечують зручну інтеграцію з рушіями, такими як Unity:

Складність узгодження:

-Виникають проблеми із синхронізацією між backend системою, діалоговими системами та іншими ігровими компонентами.

- Багато сторонніх рішень вимагають суттєвої адаптації для відповідності вимогам конкретного проєкту.

Відсутність стандартизації: Іноді інтеграція з рушієм потребує розробки додаткового функціоналу для підтримки основних механік.

Недостатня підтримка інструментів розробки та тестування

Редагування сценаріїв: Багато існуючих систем не забезпечують зручних інструментів для створення сценаріїв. Розробники змушені працювати з текстовими файлами (JSON, XML), що ускладнює їх редагування та тестування.

Автоматизація тестування: Відсутність інструментів для автоматизованого тестування виборів і сценаріїв ускладнює процес перевірки логіки гри, що може призводити до помилок у фінальному продукті.

Проблеми оптимізації продуктивності

Backend системи можуть створювати значне навантаження на систему, якщо їхня архітектура не оптимізована:

Обробка складних сценаріїв: У реальному часі системи можуть затримувати обробку виборів через великий обсяг даних і складність логіки.

Неоптимізовані підходи: Використання застарілих методів для обробки сценаріїв і виборів може призводити до зниження FPS, особливо на пристроях із низькою продуктивністю.

Висновок до розділу 1

Аналіз існуючих рішень для розробки backend систем в інтерактивних графічних пригодницьких іграх виявив низку ключових аспектів, які необхідно врахувати під час створення власної системи для нашого проєкту.

Результати аналізу

Актуальність інтерактивних ігор із нелінійним сюжетом: Сучасна ігрова індустрія активно розвивається у напрямку створення ігор, де вибори гравця впливають на розвиток подій. Це дозволяє забезпечити унікальний досвід для кожного користувача.

Особливості існуючих рішень:

- Популярні ігри жанру, такі як *Life is Strange*, *Detroit: Become Human*, *The Wolf Among Us*, демонструють ефективність використання складних сценаріїв і інтерактивності.

- У багатьох рішеннях основну роль відіграють діалогові вибори, система станів і адаптивність до дій гравців.

- Системи, такі як NodeCanvas і Dialogue System for Unity, показали високу функціональність, але також виявили свої обмеження щодо гнучкості, масштабованості та інтеграції.

Виявлені проблеми та обмеження:

- Складність масштабування великих сценаріїв із розгалуженнями.
- Відсутність адаптивності до складних взаємозалежностей між виборами.
- Недостатня кількість інструментів для редагування, тестування та візуалізації сценаріїв.
- Проблеми з інтеграцією готових рішень у специфічні проєкти.
- Оптимізація продуктивності часто залишається викликом для існуючих backend систем.

Переваги використання Unity для реалізації проєкту:
Unity забезпечує всі необхідні інструменти для створення інтерактивних сценаріїв із підтримкою нелінійного сюжету. Завдяки своїй гнучкості, легкій інтеграції та широким можливостям оптимізації, цей рушій дозволяє ефективно реалізувати всі вимоги нашого проєкту.

Подальші кроки

На основі аналізу існуючих рішень і виявлених проблем визначено основні напрями роботи:

- Розробка гнучкої архітектури backend системи, яка дозволяє легко створювати та масштабувати сценарії.
- Реалізація функціоналу для обробки виборів гравців із підтримкою складних взаємозалежностей.
- Забезпечення зручності використання.
- Інтеграція з Unity із використанням сучасних підходів до оптимізації продуктивності.

Аналіз існуючих систем і технологій дозволив чітко визначити основні вимоги до backend системи, яка буде створена, та ключові проблеми, які необхідно вирішити для досягнення поставлених цілей.

РОЗДІЛ 2
РОЗРОБКА BACKEND СИСТЕМИ

2.1 Вибір архітектури проекту

Наша backend система для управління квестами та виборами гравця побудована з урахуванням гнучкості, модульності та простоти інтеграції в ігровий рушій Unity.

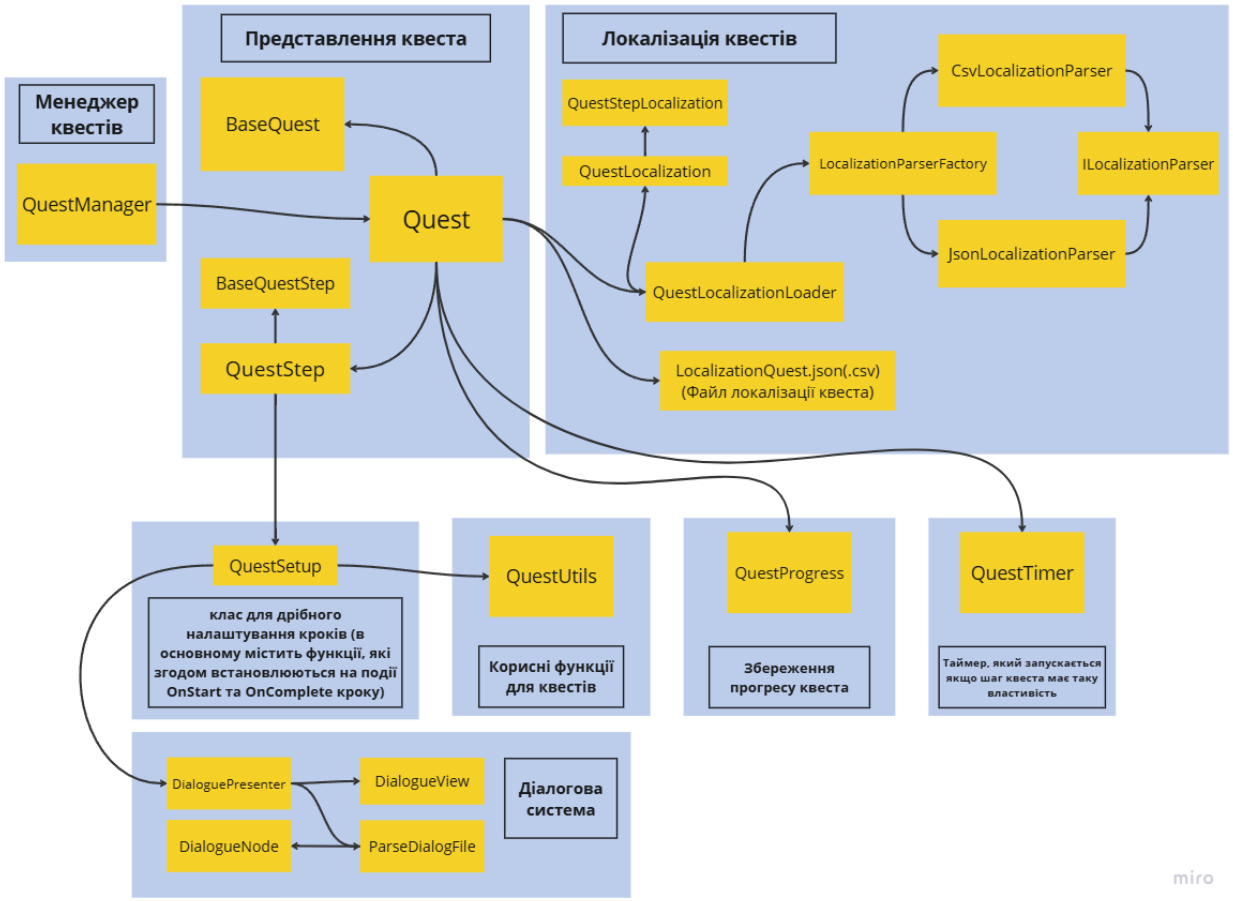

Рисунок 2.1 – Схема архітектури backend системи.

Система розділена на декілька ключових компонентів, що взаємодіють між собою:

Менеджер класів (Quest Manager)

- Відповідає за ініціалізацію, контроль та управління квестами.

- Використовується для відстеження активних квестів, їх виконання та завершення.

Представлення квесту

- Quest – головний клас, що представляє квест, містить усі його кроки (QuestStep).
- BaseQuest – базовий клас, що визначає основні характеристики, притаманні усім квестам.
- BaseQuestStep – абстрактний клас для створення кроків у квесті.
- QuestStep – конкретна реалізація кроку квесту.

Система локалізації

- QuestLocalization – містить словники для зберігання перекладів.
- QuestLocalozationLoader – завантажує локалізовані описи квестів із файлів .json або .csv.
- ILocalozationParser – інтерфейс для парсерів локалізації.
- JsonLocalozationParser, CsvLocalizationParser – класи для парсингу локалізації у відповідних форматах.
- LocalizationParserFactory – фабрика для отримання відповідного парсера, залежно від формату файлу.

Допоміжні компоненти

- QuestProgress – відповідає за збереження прогресу квесту.
- QuestTimer – реалізує таймер для кроків квесту, якщо це необхідно.
- QuestSetup – налаштовує логіку проходження квесту, наприклад підключає діалогову систему, або встановлює якісь стани, відповідно до того, як цей квест потрібен виглядати.
- QuestUtils – містить допоміжні методи для роботи з інвентарем та таймерами.

Логіка роботи системи

1. Гравець отримує новий квест, який реєструється у QuestManager.
2. Quest визначає послідовність кроків, які необхідно пройти.
3. QuestStep відповідає за виконання конкретних дій.
4. Якщо необхідно, встановлюються відповідні діалогові файли до відповідних DialoguePresenter за допомогою QuestSetup.
5. Локалізація завантажується через QuestLocalozationLoader із відповідного JSON або CSV файлу.
6. Прогрес проходження зберігається у QuestProgress, а при завершенні всіх кроків квест позначається як виконаний.

Архітектура backend системи побудована за принципом гнучкості та розширюваності. Вона дозволяє легко створювати свої нові квести, з детальним налаштуванням, змінювати їх структуру та інтегрувати локалізацію без необхідності переписування основного коду, при чому система автоматизовано розуміє який тип файлу локалізації їй передали, й відразу парсить. Використання менеджера квестів, структури кроків та локалізації дозволяє реалізувати нелінійний сюжет із різноманітними виборами гравця.

2.2 Реалізація діалогової системи

Діалогова система відповідає за відображення діалогів між персонажами гри, а також за взаємодію гравця з виборами, які можуть впливати на хід сюжету. Вона інтегрована з системою квестів, що дозволяє робити квести більш варіативними та залежними від дій гравця.

Основні завдання системи:

- Відображення текстових діалогів.
- Можливість вибору відповідей гравцем.
- Виконання дій (запуск квестів, зміни стану гри).
- Підтримка багатомовності.

Система складається з наступних основних компонентів:

- DialogueNode – окремий вузол діалогу, який містить текст репліки та можливі варіанти відповідей.
- DialoguePresenter – клас, який керує відображення діалогів, взаємодіє з UI та передає дані для відображення.
- ParseDialogFile – парсер діалогових файлів, який завантажує структуру діалогів із переданих xml файлів.
- DialogueView – безпосередньо відповідає за візуалізацію діалогу на екрані.

Взаємодія діалогової системи з квестами

Діалогова система використовується для:

- **Запуску квестів через діалоги** – коли гравець взаємодіє з NPC, може бути ініційований новий квест через віповідний вибір.
- **Прогресування квесту** - виконання певних квестових завдань може потребувати розмов з NPC.
- **Вибору варіантів відповіді** – деякі квести мають нелінійну структуру, яка залежить від вибору гравця в діалогах.

Структура діалогового файлу

Файл збережений у форматі XML, що дозволяє легко змінювати та додавати нові діалоги. Розглянемо основні елементи на прикладі (див. рис. 2.2).

Діалог завантажується у систему через парсер XML (ParseDialogFile), який перетворює його у внутрішню структуру DialogueNode.

Основні етапи роботи:

- Гравець ініціює діалог з NPC.
- Парсер завантажує XML, передає DialoguePresenter перший DialogueNode.
- Гравець обирає варіант відповіді – система обирає наступний вузол.

- Якщо вузол містить <action id>, QuestManager може активувати відповідний квест, якщо цей <action id> прив’язаний до ініціалізації квесту.

```
<dialog>
  <name>
    <ru>Синий Куб</ru>
    <ua>Синій Куб</ua>
    <uk>Blue Cube</uk>
  </name>
  <message>
    <ru>О, ты вернулся! Ты принес три яблока?</ru>
    <ua>О, ти повернувся! Ти приніс три яблука?</ua>
    <uk>Oh, you're back! Did you bring the three apples?</uk>
  </message>
  <voice>
    <ru>returnedWithApplesRu</ru>
    <ua>returnedWithApplesUa</ua>
    <uk>returnedWithApplesUk</uk>
  </voice>

  <dialog>
    <name>
      <ru>Персонаж</ru>
      <ua>Персонаж</ua>
      <uk>Character</uk>
    </name>
    <message>
      <ru>Да, вот они.</ru>
      <ua>Так, ось вони.</ua>
      <uk>Yes, here they are.</uk>
    </message>
    <voice>
      <ru>hereApplesRu</ru>
      <ua>hereApplesUa</ua>
      <uk>hereApplesUk</uk>
    </voice>

    <dialog>
      <name>
        <ru>Синий Куб</ru>
        <ua>Синій Куб</ua>
        <uk>Blue Cube</uk>
      </name>
      <message>
        <ru>Замечательно! Спасибо тебе большое. Это поможет мне в моей работе. Вот твоя награда!</ru>
        <ua>Чудово! Дуже тобі дякую. Це допоможе мені в моїй роботі. Ось твоя нагорода!</ua>
        <uk>Wonderful! Thank you so much. This will help me in my work. Here's your reward!</uk>
      </message>
      <voice>
        <ru>thanksForApplesRu</ru>
        <ua>thanksForApplesUa</ua>
        <uk>thanksForApplesUk</uk>
      </voice>
      <action id="5"></action>
    </dialog>
  </dialog>
</dialog>
```

Рисунок 2.2. – приклад діалогового xml-файлу

Взаємодія діалогів із квестами

Якщо вибір гравця в діалозі містить тег <action id>, то система виконує пов’язану дію (наприклад активує квест, переходить до іншого шагу, змінює стан якогось предмета або все це в купі).

Діалогова система побудована за принципом гнучності та маштабованості. Використання XML дозволяє легко додавати нові діалоги та підтримувати багатомовність. Інтеграція з QuestManager забезпечує взаємозв’язок між діалогами та виконанням завдань у грі.

2.3 Реалізація системи квестів

Система квестів у нашій backend-системі відповідає за ініціалізацію, виконання та завершення місій у грі. Вона забезпечує можливість додавання основних та побічних місій, відстеження статусу завдань, а також інтерактивну взаємодію з іншими системами гри.

Реалізація цієї системи дозволяє гравцеві виконувати місії, отримувати винагороди, змінювати стан гри, а також впливає на розвиток сюжету через нелінійні квестові завдання.

Ініціалізація квестів

Кожна місія має назву, опис, нагороду, список завдань та умови завершення. В момент, коли місія стає доступною для гравця (наприклад, через діалог із NPC), вона додається у список активних місій.

Ініціалізація місії виконується через QuestManager, який зберігає стан та контролює виконання її завдань.

Приклад коду для додавання нової місії:

```
Quest newQuest = new Quest("Допомога мандрівнику", "Знайди три яблука та
поверни їх NPC.");
QuestManager.AddQuest(newQuest);
```

Виконання квестів

Кожна місія складається з кроків, які гравець повинен виконати. Система передбачає різні типи кроків місій, наприклад:

- Збір предметів (CollectItemQuestStep) – знайти певну кількість предметів.

- Розмова з NPC (TalkToQuestStep) – необхідно поспілкуватися з певним персонажем.

- Обмеження за часом (QuestTimer) – модифікатор, додається коли крок повинен бути виконан у визначений період.

У момент виконання кожного кроку квесту система перевіряє його умови та змінює статус місії. Якщо всі кроки виконані, місія завершується.

Приклад реалізації кроку збору предметів:

```
public class CollectItemQuestStep : QuestStep
{
    private string _itemName;
    private int _requiredAmount;
    private int _currentAmount = 0;

    public CollectItemQuestStep(string itemName, int amount)
    {
        _itemName = itemName;
        _requiredAmount = amount;
    }

    public void OnItemCollected(string itemName)
    {
        if (itemName == _itemName)
        {
            _currentAmount++;
            if (_currentAmount >= _requiredAmount)
            {
                CompleteStep();
            }
        }
    }
}
```

Загалом, кроки можна налаштовувати самому дуже детально, але якщо потрібен якийсь шаблон кроку, тому що він повторюється дуже часто, то можна зробити клас цього типу кроку, та додавати їх до загального квесту.

Для детального налаштування кроків місії використовується клас QuestSetup. Він дозволяє гнучко задавати поведінку кожного кроку, додаючи умови початку, завершення та взаємодії з ігровими системами. У ньому зберігаються функції, які встановлюються на події OnStart та OnComplete кожного кроку. Це дає змогу налаштовувати специфічні сценарії, наприклад вимагати виконання певних дій до завершення кроку або запускати додаткові механіки у відповідь на його завершення. Приклад функцій початку та кінця кроку:

```
public void SetupFirstStep(QuestStep step)
{
    _dialogPresenter._dialoguesArr = new TextAsset[]{
    Resources.Load<TextAsset>("Dialogs/dialogQuestStart") };

    _onDialogEndHandler = actionId = >
```

```
        {
            if (actionId.HasValue)
            {
                step.SetComplete(true);
                _quest.CompleteStep(actionId.Value);
            }
        };

        _dialogPresenter.OnDialogEnd += _onDialogEndHandler;
    }

    public void FinishFirstStep()
    {
        _dialogPresenter._dialoguesArr = new TextAsset[]{
        Resources.Load<TextAsset>("Dialogs/dialogQuestAwait") };
        _dialogPresenter.OnDialogEnd -= _onDialogEndHandler;
        _onDialogEndHandler = null;
    }
```

Допоміжні інструменти, які допомагають інтегрувати систему місій з іншими механіками гри, реалізовані у QuestUtils. Цей клас містить набір корисних методів для роботи з квестами, таких як перевірка статусу виконання кроку, взаємодія з інвентарем, перевірка стану NPC або об’єктів у ігровому світі. Ці інструменти налаштовуються розробниками в залежності з якими ігровими системами квести повинні взаємодіяти.

Приклад функції перевірки чи є потрібний нам предмет в необхідній кількості в нашому інвентарі:

```
    public void CheckItemOnInventory(int idItem, int countItem, Action onFound)
    {
        List<ItemData> items = _inventory.GetAllItems();
        foreach(ItemData item in items)
        {
            if (item.Id == idItem && item.CountItem == countItem)
            {
                onFound ? .Invoke();
            }
        }
    }
```

Завдяки QuestSetup та QuestUtils наша система місій стає гнучкою, забезпечуючи можливість створювати складні квести з різними умовами виконання та інтегрувати їх з іншими механіками гри без зайвих змін у головній логіці квестів.

Завершення квестів

Після виконання всіх кроків місій викликається метод OnQuestComplete(), який змінює статус місії, видає нагороду гравцеві та оновлює стан гри.

Функція завершення квесту:

```
public void OnQuestComplete()
{
    GiveReward();
    QuestManager.MarkQuestAsCompleted(this);
}
```

Взаємодія місій з іншими системами

Діалогова система – місії можуть бути отримані або завершені через діалоги з NPC.

Інвентар – виконання місій може включати збір предметів, тому легко можна підключити інвентар до системи квестів, якщо такий інвентар реалізований в грі.

Таймер – деякі квести мають обмеження за часом, якщо гравець не встигає виконати умови проходження кроку за деякий час, то крок вважається проваленим.

Збереження прогресу – стан виконаних місій зберігається, щоб гравець міг повернутися до них після перезапуску гри.

Отже, реалізована система місій, яка дозволяє гнучко створювати різні типи завдань, контролювати їхній прогрес та забезпечувати нелінійність сюжету. Інтеграція з іншими механіками гри дає змогу створювати цікаві та варіативні місії, що покращує ігровий досвід.

2.4 Реалізація системи локалізації

Система локалізації у нашій backend-системі дозволяє відображати текстові дані (назви квестів, описи завдань, діалоги) відповідно до обраної мови гри. Вона підтримує роботу з декількома мовами, завантаження локалізованих даних із файлів, а також інтеграцію з іншими системами гри.

Локалізація ж важливим аспектом для створення адаптивного ігрового середовища, що дозволяє користувачам сприймати інформацію зручною для них мовою.

Формат збереження локалізації

Локалізовані дані зберігаються у форматах JSON або CSV для гнучкості та зручності редагування.

Приклад CSV-файлу:

```
Id, en, ru, ua
0, "Talk to the NPC to find out the details of the quest", "Поговорите с нпс чтобы узнать детали задания", "Поговоріть з NPC, щоб дізнатися подробиці завдання"
1, "Find 3 apples", "Найдите 3 яблока", "Знайдіть 3 яблука"
1, "Find 3 matchboxes", "Найдите 3 коробки спичек", "Знайдіть 3 коробки сірників"
2, "Find and talk to the friend of the blue cube", "Найдите и поговорите с другом синего куба", "Знайдіть і поговоріть з другом синього куба"
3, "Return to the blue cube and give the apples", "Вернитесь к синему кубу и отдайте яблоки", "Поверніться до синього куба і віддайте яблука"
4, "Return to the blue cube and report that you found his friend", "Вернитесь к синему кубу и сообщите о том что вы нашли его друга", "Поверніться до синього куба і повідомте, що ви знайшли його друга"
5, "Mission completed", "Миссия завершена", "Місія завершена"
6, "Mission failed", "Миссия провалена", "Місія провалена"
```

Приклад JSON-файлу:

```
"QuestName": {
    "en": "Help the Blue Cube",
    "ru" : "Помощь синему кубу",
    "ua" : "Допомога синьому кубу"
},
"Steps" : [
{
    "Id": 0,
    "Description" : {
        "en": "Talk to the NPC to find out the details of the quest",
        "ru" : "Поговорите с нпс чтобы узнать детали задания",
        "ua" : "Поговоріть з NPC, щоб дізнатися подробиці завдання"
    }
}
```

Реалізація локалізації

Локалізація реалізована через гнучку систему завантаження та обробки текстових даних, що дозволяє підтримувати кілька мов та легко змінювати їх без змін у коді гри.

Для цього використовується механізм парсингу локалізаційних файлів, який автоматично розпізнає формат (JSON або CSV), обробляє дані та зберігає їх для швидкого доступу. Фабрика парсерів обирає потрібний механізм для роботи з конкретним файлом, що дозволяє легко додавати підтримку нових форматів у майбутньому. Якщо команда розробників вирішить

використовувати інші формати збереження локалізації, то їм не буде потрібно переписувати весь код, а тільки додати новий парсер, який буде парсити новий формат локалізації, та оновити функцію автоматичного вибору механізму парсингу згідно з типом файлу.

Усі локалізовані дані зберігаються у вигляді словника, де кожен запис містить унікальний ідентифікатор та відповідні локалізовані тексти. Завдяки такому підходу ми можемо отримувати необхідний текст у будь-який момент, просто звернувшись до системи локалізації.

Зміна мови здійснюється динамічно – при зміні налаштувань гри система перезавантажує всі текстові дані та оновлює інтерфейс без потреби перезапуску. Це забезпечує зручність для користувача, а також спрощує тестування локалізації безперервно під час розробки.

Завантаження локалізованих даних

Коли гра запускається, QuestLocalizationLoader завантажує всі доступні локалізовані тексти.

Приклад коду завантаження JSON-файлу:

```
public class JsonLocalizationParser : ILocalizationParser
{
    public QuestLocalization Parse(string content)
    {
        return JsonConvert.DeserializeObject<QuestLocalization>(content);
    }
}
```

Приклад коду для отримання локалізованого тексту:

```
public string GetStepDescription(int stepId, string language)
{
    var steps = _questLocalization.Steps.Where(s = > s.Id ==
stepId).ToList();

    var descriptions = steps
        .Where(s = > s.Description.ContainsKey(language))
        .Select(s = > s.Description[language])
        .ToList();

    return descriptions.Any() ? string.Join("\n", descriptions) :
"Description not found";
}
```

Динамічне оновлення локалізації

Гравець може змінити мову гри у налаштуваннях. У такому випадку всі текстову елементи повинні оновитися без перезапуску гри.

Приклад функції оновлення мови:

```
private void UpdateLanguage()
{
    List<QuestStep> result = new List<QuestStep>();

    foreach(var step in _steps)
    {
        if (step.ActionId == CurrentStepIndex)
        {
            result.Add(step);
        }
    }

    GetLocalizationLanguage();
    UpdateStepDescription(result);
}
```

Реалізована система локалізації дозволяє гнучко керувати мовою квестів гри, підтримує завантаження текстів із різних форматів та легко інтегрується з іншими системами.

2.5 Інтеграція Zenject для управління залежностями

У нашому проєкті використовується Zenject – Dependency Injection (DI) фреймворк для Unity, що дозволяє ефективно керувати залежностями між об’єктами та покращує архітектуру проєкту.

Використання Zenject забезпечує наступні переваги:

- Ослаблення зв’язків між компонентами – клас не створюють екземпляри інших класів напряму, а отримують їх через ін’єкцію залежностей.
- Полегшене тестування – можна легко підміняти залежності під час тестування.
- Гнучке управління життєвим циклом об’єктів – контроль створення та знищення об’єктів через Zenject Container.

Використання Zenject у нашій системі

Zenject інтегрується через Installer, який конфігурує залежності та реєструє їх у контейнері.

Наприклад, реєстрація менеджера квестів (QuestManager) як синглтона:

```
public class GameInstaller : MonoInstaller
{
    public override void InstallBindings()
    {

Container.BindInterfacesAndSelfTo<GameSettingsRepository>().AsSingle();
    }
}
```

Тепер його можна отримати у будь-якому іншому класі без створення його вручну, а тільки зробив ін’єкцію:

```
[Inject]
public void Construct(QuestManager questManager)
{
    _questManager = questManager;
}
```

Кожен квест у нашій системі може мати зовнішні залежності, такі як інвентар, діалогова система або система взаємодії з навколишнім світом. Завдяки Zenject, ці об’єкти передаються автоматично:

```
[Inject]
public Quest(Inventory inventory, DialogueSystem dialogueSystem)
{
    _inventory = inventory;
    _dialogueSystem = dialogueSystem;
}
```

Це дозволяє легко замінювати залежності, наприклад, під час тестування або розширення функціоналу. Отже використання Zenject у нашому проєкті значно покращує архітектуру квестової системи, зменшуючи жорсткі залежності між об’єктами та полегшуючи масштабування та тестування. Завдяки гнучній DI-системі, ми можемо легко замінювати компоненти, додавати нові механіки та ефективно управляти життєвим циклом об’єктів.

Висновок до розділу 2

У цьому розділі було описано реалізацію основних компонентів системи: архітектуру, діалогову систему, систему квестів, сценаріїв та локалізації. Показано, як побудована структура класів, як відбувається обробка виборів користувача, а також яким чином дані локалізації підключаються до гри.

РОЗДІЛ 3
ТЕХНІЧНА РЕАЛІЗАЦІЯ

3.1 Використання Unity та його функціоналу

Unity – це один із найпопулярніших рушіїв для створення ігор, що забезпечує широкий набір інструментів для реалізації складних ігрових механік, включаючи системи анімації, фізики, навігації та інтеграцію з backend-системами. У нашому проєкті Unity використовується як основний рушій, на базі якого реалізовано систему управління квестами, сценаріями та діалогами.

Система компонентів та об’єктно-орієнтований підхід

Unity використовує компонентну архітектуру, де кожен об’єкт у сцені може мати набір різних компонентів, що визначають його поведінку. У нашому проєкті квести, NPC, інвентар та інші ігрові механіки реалізовані через MonoBehaviour-компоненти.

Наприклад QuestManager є компонентом, що керує активними квестами:

```
public class QuestManager : MonoBehaviour
{
    private List<Quest> _quests;
    private bool _isQuestRun;
    private int _currentQuestIndex;

    private void RunQuest(int questIndex)
    {
        _currentQuestIndex = questIndex;
        _quests[_currentQuestIndex].OnComplete += QuestOnComplete;
        _quests[_currentQuestIndex].RunQuest();
        _isQuestRun = true;
    }
}
```

Система подій Unity

Unity дозволяє використовувати події та делегати для комунікації між різними системами без жорстких зв’язків. У нашій backend-системі ми активно використовуємо події для відстеження змін у стані квестів, діалогів і виборів гравця.

Наприклад:

```
public event Action OnQuestCompleted;

private void CompleteQuest()
{
    OnQuestCompleted ? .Invoke();
}
```

Робота з UI

Unity надає Canvas та UI-компоненти для створення інтерфейсу. У нашому проєкті ця система використовується для відображення діалогових вікон, опису квесту та виборів гравця.

UI оновлюється у реальному часі відповідно до прогресу гравця в системі місій:

```
public void UpdateQuestUI(Quest quest)
{
    questTitleText.text = quest.Title;
    questDescriptionText.text = quest.Description;
}
```

Збереження прогресу (PlayerPPrefs, JSON, ScriptableObjects)

Щоб зберігати стан квестів, виборів гравця та діалогів, у проєкті використовується серіалізація JSON та ScriptableObjects. Це дозволяє легко зберігати дані та завантажувати їх при повторному запуску гри:

```
public void SaveProgress()
{
    var progress = new QuestProgress
    {
      QuestName = _questName,
      CurrentStepIndex = _currentStepIndex
    };

    string json = JsonUtility.ToJson(progress);
    PlayerPrefs.SetString($"QuestProgress_{_questName}", json);
    PlayerPrefs.Save();
}
```

Фізичний рушій Unity (Physic 3D/2D)

У випадку інтерактивних об’єктів, які є частиною місій (наприклад, предмети для збору або двері, що відкриваються після виконання квесту), використовується Physics.Raycast(), Rigidbody та Trigger-колайдери:

```
if (Physics.Raycast(ray, out hit))
{
```

```
        if (hit.collider.CompareTag("QuestItem"))
        {
            CollectItem(hit.collider.gameObject);
        }
    }
```

3.2 Модулі проєкту та їх функціонал

У нашій системі модульна архітектура відіграє ключову роль, оскільки забезпечує гнучкість, розширюваність та легкість підтримки коду. Кожен модуль відповідає за конкретну частину логіки гри, що дозволяє легко модифікувати та розширювати систему без ризику порушення її загальної роботи.

Основні модулі нашого проєкту

Менеджер квестів (QuestManager)

- Відповідає за загальне управління квестами в грі.
- Відстежує активні, завершені та невиконанні квести.
- Передає інформацію іншим модулям про зміну стану квестів.
- Використовує подійну систему Unity для взаємодії з іншими компонентами.

Система квестів (Quest, QuestStep, QuestSetup)

- Описує структуру кожного квесту, його кроки та умови завершення.
- QuestStep визначає конкретні дії, які гравець має виконати.
- QuestSetup використовується для налаштування унікальних механік окремих квестів (умов виконання кроку, встановлення діалогів чи збору предметів)

Система діалогів (Dialogue System)

- Обробляє структуровані діалогові файли (XML).
- Визначає варіанти вибору гравця.
- Інтегрується із системою квестів для оновлення статусу місій на основі вибору гравця.

Збереження прогресу (QuestProgress)

- Відповідає за збереження та відновлення стану квестів.
- Дозволяє гравцям продовжувати виконання місій після перезапуску гри.
- Використовує JSON-серіалізацію для збереження інформації у PlayerPrefs.

Система локалізації (QuestLocalization, QuestLocalizationLoader)

- Завантажує локалізовані тексти для квестів, діалогів та інших текстових елементів.
- Використовує гнучку структуру парсерів для роботи з JSON та CSV.
- Дозволяє змінювати мову гри без перезапуску.

Система таймерів (QuestTimer)

- Дозволяє встановлювати обмеження по часу для виконання певних квестів.
- Використовується для створення часових квестів або обмежених у часі завдань.
- Інтегрується із загальною логікою QuestStep.

Усі модулі проекту тісно інтегровані між собою, але водночас залишаються слабко зв’язаними завдяки використанню подій, фабрик та DI-підходу через Zenject.

3.3 Оптимізація продуктивності проекту

Оптимізація продуктивності є ключовим аспектом розробки ігрових систем. У нашому проекту застосовано ряд ефективних методів оптимізації, що дозволяють мінімізувати використання ресурсів, забезпечити стабільну роботу логіки та плавний ігровий процес.

Оптимізація обробки квестів та сценаріїв

Оскільки наша система може містити велику кількість квестів, виборів та сценаріїв, важливо мінімізувати витрати ресурсів на їхню обробку.

Подієва система

- Замість постійного опитування (Update()), система квестів реагує на події (наприклад, завершення кроку квесту або зміна стану NPC)
- Використання делегатів та подій значно зменшує навантаження на процесор

```
public class QuestStep
{
    public event Action OnStepCompleted;

    public void CompleteStep()
    {
        OnStepCompleted ? .Invoke();
    }
}
```

Ліниве завантаження

- Квести завантажуються в момент їхньої активації, а не всі одразу при старті гри.
- Це дозволяє економити пам’ять у великих ігрових світах.

Оптимізація локалізації

Локалізація реалізована через попереднє кешування текстів, що зменшує навантаження при кожному зверненні до файлів локалізації.

- Локалізовані дані зберігаються в словнику, що дозволяє швидко отримувати доступ до перекладів без повторного зчитування файлів.

```
private Dictionary<string, string> localizationCache;

public string GetLocalizedText(string key)
{
    return localizationCache.TryGetValue(key, out var text) ? text : "Missing
Text";
}
```

Збереження та завантаження прогресу

Збереження прогресу реалізовано через JSON-серіалізацію, що дозволяє ефективно зберігати та відновлювати стан квестів без зайвих обчислень.

- Автоматичне стиснення даних перед збереженням (використовується мінімізація JSON для зменшення обсягу файлів)

```
string json = JsonConvert.SerializeObject(progressData, Formatting.None);
File.WriteAllText(savePath, json);
```

Оптимізація роботи з Unity

Unity має ряд обмежень у продуктивності, тому важливо уникати неоптимальних практик.

Мінімізація Update()

- Використовуються корутини для рідкісних оновлень замість постійного виконання коду в Update().

```
private IEnumerator Timer(float duringTimer)
{
    TMP_Text timer =
gameObject.transform.GetChild(1).GetComponent<TMP_Text>();
    timer.text = duringTimer.ToString();
    float time = duringTimer;

    for (int i = 0; i < duringTimer; i++)
    {
        yield return new WaitForSeconds(1);
        time--;
        timer.text = time.ToString();
    }

    var quest = _questManager.FindActiveQuest();
    quest.CompleteStep(6);
}
```

Завдяки ефективній оптимізації ми змогли мінімізувати витрати пам’яті, зменшити навантаження на процесор та забезпечити плавний ігровий процес. Це дозволяє системі масштабуватися та бути гнучкою для майбутній доповнень без значних втрат продуктивності.

Висновок до розділу 3

У цьому розділі було розглянуто використання функціоналу Unity для реалізації backend системи, описано модулі проекту та їх призначення. Окрему увагу приділено оптимізації продуктивності – зменшенню навантаження на систему шляхом заміни постійних перевірок на подійний підхід.

РОЗДІЛ 4
РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНОЇ ЧАСТИНИ

4.1 Тестування backend-системи

Метою тестування є перевірка коректності роботи основних механік системи, включаючи:

- Реєстрацію та виконання квестів.
- Відстеження стану виконання кроків.
- Взаємодію з іншими системами гри (наприклад, отримання предметів).
- Перевірку стабільності виконання квестів у межах ігрового процесу.

Функціональне тестування

На цьому етапі перевіряється, чи правильно працює система квестів та чи коректно змінюються їхні стани при виконанні умов.

Тестування створення та ініціалізації квесту

Опис: Перевіряємо, чи створюється квест із заданими параметрами та чи додається до списку квестів.

Дії: Використовуємо редактор Unity для створення об’єкта квесту та перевіряємо його ініціалізацію.

Очікуваний результат: Квест повинен бути створений із заданими параметрами (назва, опис, список кроків).

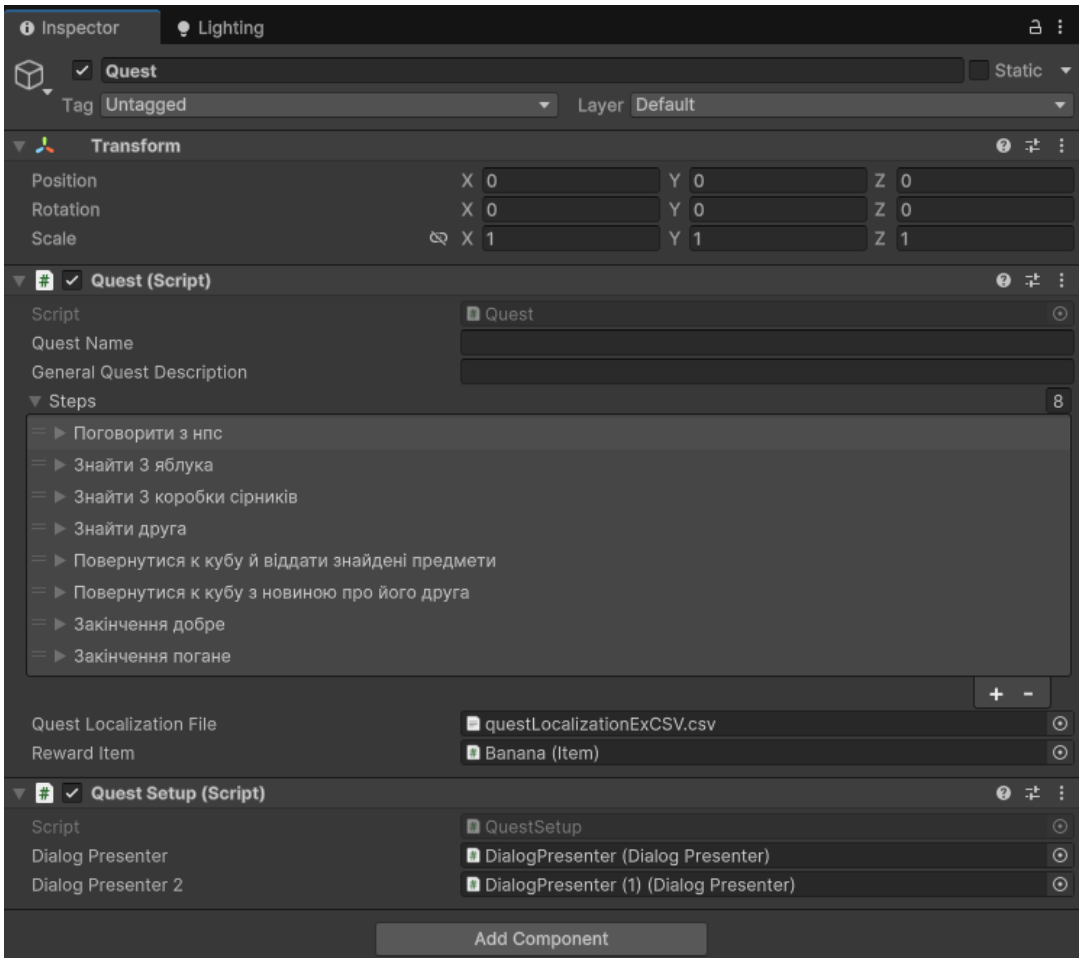

Рисунок 4.1 – Об’єкт квесту в інспекторі

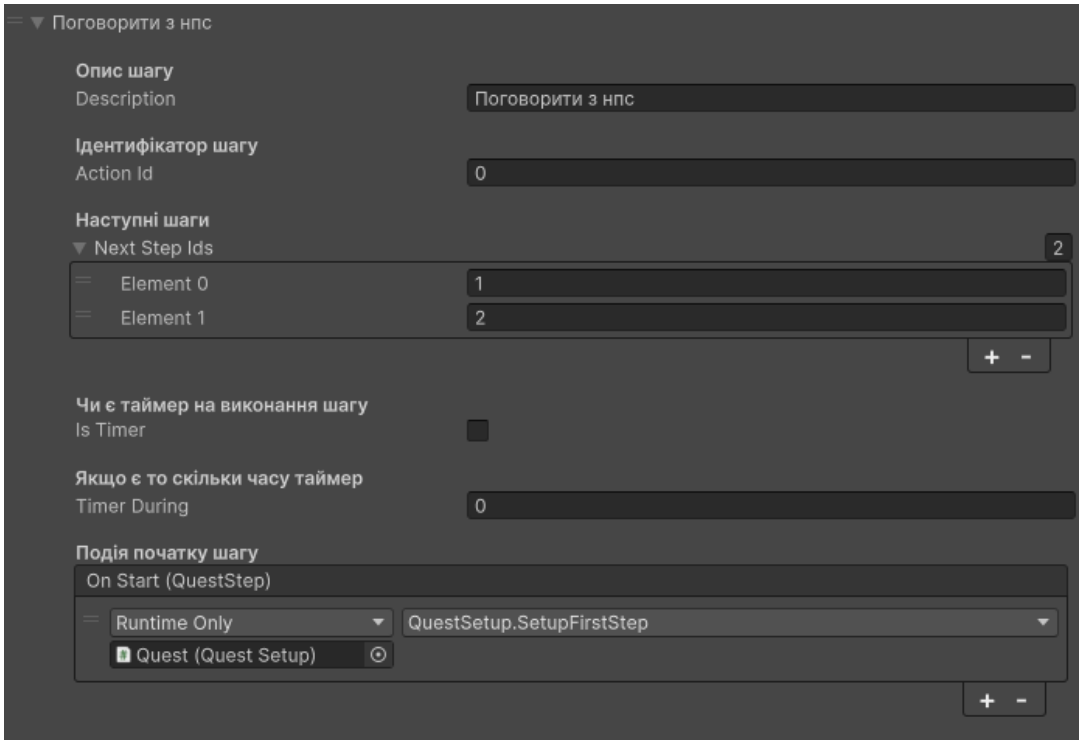

Рисунок 4.2 – Приклад структури заповнення кроку квесту

Рисунок 4.3 – Результат реєстрації квесту

Як бачимо, квест при запуску гри в нас зареєструвався, й одразу запустив перший крок до дії. Опис який ми вписуємо в інспекторі не так важливий, це більше зроблено для того, щоб відрізняти кроки один від одного, а локалізація береться з відповідно завантаженого файлу локалізації. Як висновок можемо сказати, що реєстрація квесту проходить успішно.

Тестування оновлення стану квесту

Опис: Квест повинен змінювати свій стан відповідно до того як виконуються кроки.

Дії: Виконати кожен крок квесту.

Очікуваний результат: При виконанні кожного кроку він має переходити до наступного, а після останнього – весь квест завершується.

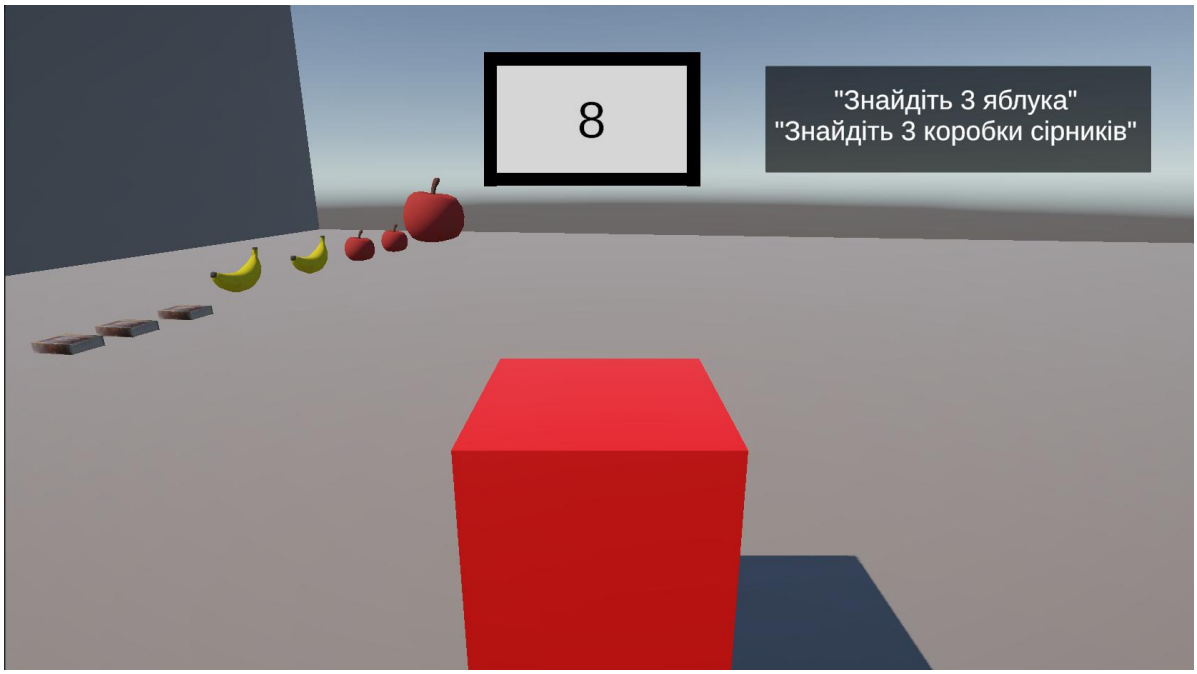

Рисунок 4.4 – Наступний крок квесту

Після того як ми поговорили з NPC, ми взяли в нього завдання на знаходження предметів, й після діалогу в нас минулий крок завершився, а наступний почався. Як можемо побачити, він навіть має таймер що запустився, тому заодно ми побачили що функція обмеження в часі для кроків також працює відмінно.

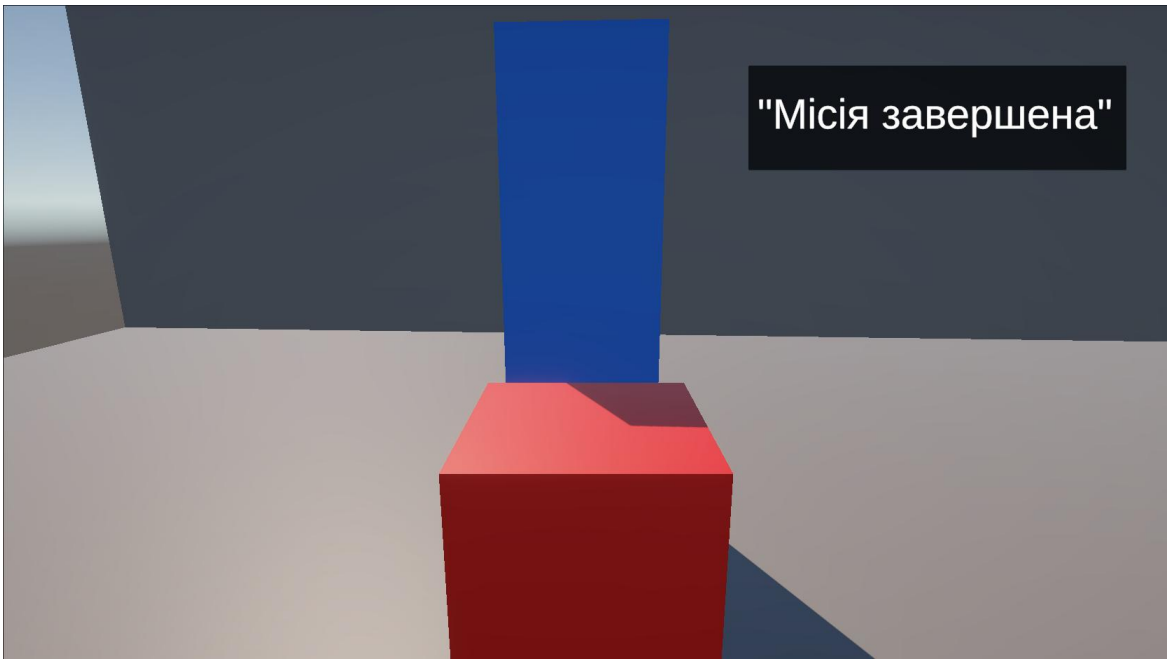

Рисунок 4.5 – Завершення квесту

Після того як ми виконали всі кроки цього квесту, він позначається як пройдений.

Тестування нелінійності

Опис: Квест повинен піти по іншій гілці розвитку, якщо зробити відповідні дії в моменті розвилки.

Дії: Обрати інший варіант відповіді в діалозі.

Очікуваний результат: В нас запуститься не крок зі збором предметів, а інший.

Рисунок 4.6 – Варіанти відповідей

Ми повинні обрати «Знайти друга», й після цього в нас квест піде по іншій сценарній гілці.

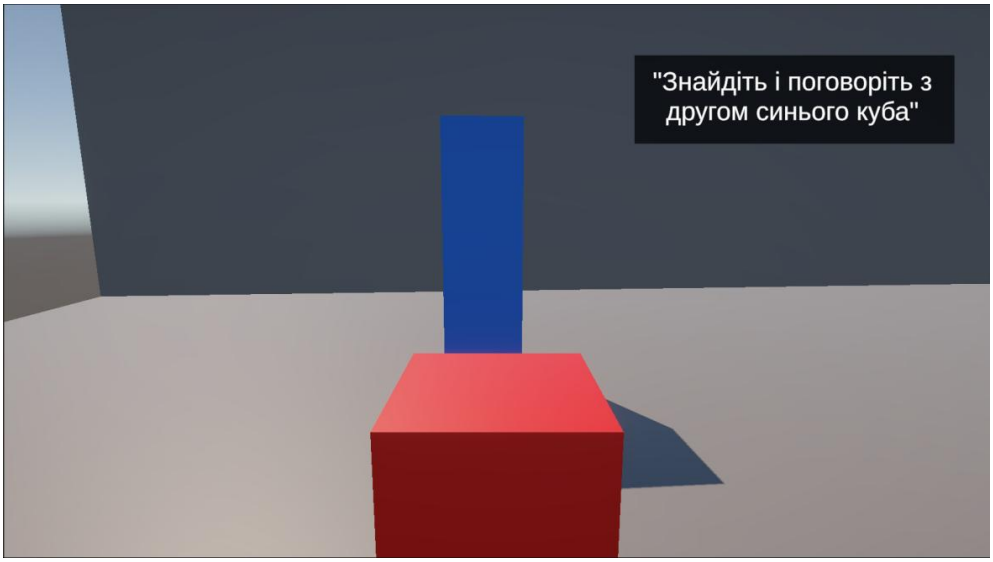

Рисунок 4.7 – Нелінійна зміна кроку

Як можемо побачити, в нас запустилась інша сценарна гілка, й по кроку ми повинні знайти друга синього куба, а не 3 яблука. Тобто, як висновок, можна сказати що нелінійність в нашій системі працює.

Тестування збереження прогресу

Опис: Квест зберігає свій стан при виході з гри, й при перезавантаженні він починається з того місця де ми закінчили.

Дії: Пройти квест до половини, вийти з гри, та зайти в неї заново.

Очікуваний результат: Квест продовжиться з того місця де ми вийшли.

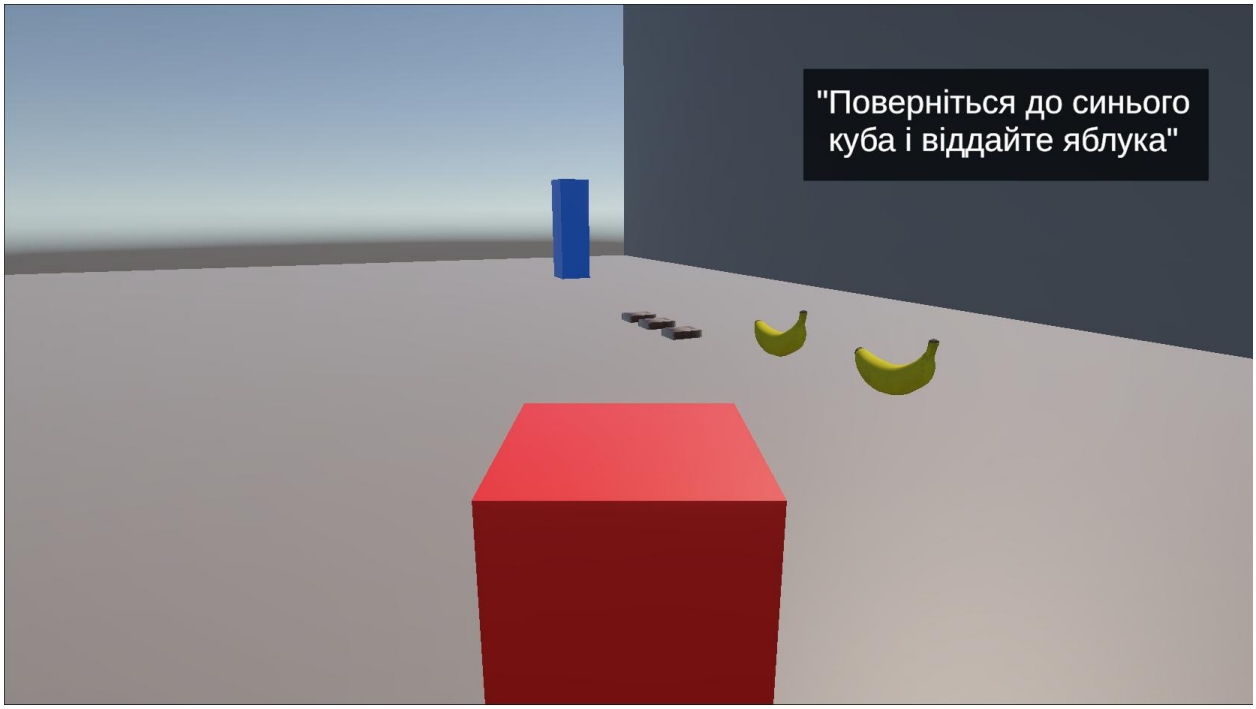

Рисунок 4.8 – Пройдена половина квесту

З цього моменту ми закінчуємо гру, після чого запускаємо її заново, й повинні побачити, що гра почнеться з того місця, де ми вийшли, а якщо точніше з кроку, який був активний тоді, коли ми вийшли.

Рисунок 4.9 – Результат перезавантаження гри

Як можемо побачити, з’явились ми не на тому ж місці, бо цей функціонал не відноситься до нашої системи, але крок, який був активним при виході, так і залишився активним.

4.2 Аналіз продуктивності

Мета аналізу

Основною метою аналізу продуктивності є перевірка того, як система обробляє зміну стану квесту в реальному часу та як це впливає на використання ресурсів гри. Важливо оцінити навантаження, яке створює система квестів, та виявити можливі вузькі місця, що можуть призвести до падіння FPS або затримок в оновленні даних.

Методика аналізу

Оцінка продуктивності виконувалась у середовищі Unity з використанням Profiler. Було протестовано такі сценарії:

- Сценарій 1: Звичайне функціонування квестової системи (один активний квест в спокійному темпі)

- Сценарій 2: Активна взаємодія (швидке виконання кількох кроків підряд)
- Сценарій 3: Аналіз оновлення стану квесту в реальному часу через Update()

Наша система квестів, як було зазначено раніше, використовує підхід подійної системи, яка використовує ресурси комп’ютера тільки в той момент коли це необхідно. Але існує й інший підхід, такий як реалізовувати перевірку станів через Update(), тобто перевіряти стан квесту кожен кадр, що значно знижує продуктивність системи. Для наглядності було протестовано обидва підходи, щоб продемонструвати важливість оптимізації.

Сценарій 1

Рисунок 4.10 – графік використання CPU в Profiler при Сценарії 1

Як можемо побачити, при спокійному проходженні квесту в нас графік рівний, з рідкими незначними скачками, що доволі гарно сказується на продуктивності системи. Й такий графік на протязі усього тестування. Графік відображає навантаження на CPU саме роботи скриптів.

Рисунок 4.11 – час виконання скриптів в спокійному режимі

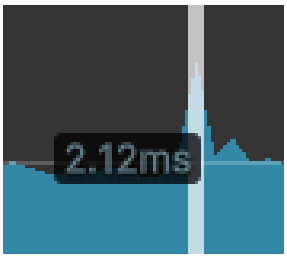

Рисунок 4.12 – час виконання скрипту оновлення стану місій

Тобто як можемо побачити, в спокійному режимі саме складне оновлення зайняло 2.12 мс.

Сценарій 2

Рисунок 4.13 – графік використання CPU в Profiler при Сценарії 2

Аналізуючи другий графік можна сказати, що швидкість виконання квесту майже не впливає на продуктивність системи, що показує її стабільність, так як цей графік схожий майже 1 в 1 з графіком Сценарію 1.

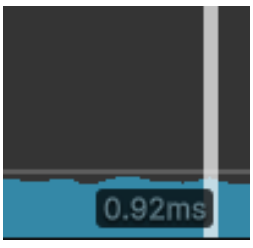

Рисунок 4.14 – час виконання скриптів в швидкому темпі

Рисунок 4. 15 – час виконання скрипту оновлення стану квесту в швидкому темпі

Отже, при швидкому виповненні квесту час виконання скриптів не значно підвищився, й складає 0.92 мс в стані підтримання функціонування гри, та 2.17 мс при оновленні стану квеста.

Сценарій 3

Для того, щоб протестувати третій сценарій, потрібно трішки споганити нашу систему, переробивши її під перевірку стану квесту кожен кадр. Для цього виконаємо наступні дії:

Зробимо оновлення опису квесту кожен кадр:

```
void Update()
  {
    _questDescriptionPanel.SetActive(true);
    _questDescriptionPanel.transform.GetChild(0).GetComponent<TMP_Text>
().text = description;
  }
```

Перевірка умови завершення кроку кожен кадр:

```
public void SetupFirstStep(QuestStep step)
  {
    _dialogPresenter._dialoguesArr = new TextAsset[] {
Resources.Load<TextAsset>("Dialogs/dialogQuestStart") };
    _isStepCompleted = false;
    _actionId = null;
  }

  private void Update()
  {
    if (!_isStepCompleted && _dialogPresenter.IsDialogEnded(out int
actionId))
```

```
{
  _isStepCompleted = true;
  _actionId = actionId;
  CompleteStep();
}}
```

Друга дія також буде застосована не тільки до перевірки умови першого кроку, а й всіх інших.

Рисунок 4.16 – графік використання CPU в Profiler при Сценарії 3

Як можемо побачити, навантаження на процесор значно підвищився, за рахунок того що обчислення одного й того ж виконується кожен кадр.

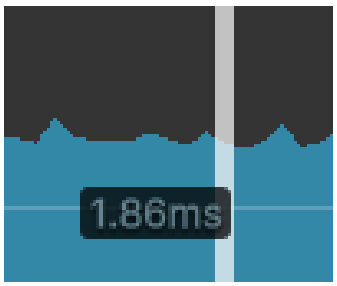

Рисунок 4.17 – час виконання скриптів через Update

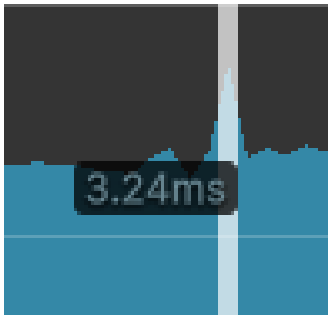

Рисунок 4.18 – час виконання найбільш затратного скрипта через Update

Отже, якщо використовувати такий не дуже продуктивний підхід, то час виконання скриптів виростає до 1,86 мс, що є приблизно в 2 раза більше, ніж при використанні подійного підходу, а при оновленні стану квесту до 3,24 мс.

Результати тестування

Використання ресурсів при стандартному функціонуванні квестової системи: в середньому оновлення стану займає 2,13 мс, що є мінімальним навантаженням.

Вплив швидкої взаємодії з квестами: коли гравець швидко виконує кілька дій підряд, навантаження на CPU незначно збільшується до 0,05 мс.

Вплив системи оновлення стану квесту: використання Update() підходу створює зайве навантаження, тому що перевірка відбувається на кожному кадрі й збільшує навантаження на систему приблизно в 2 раза.

4.3 Вплив оптимізації на продуктивність

Проведені оптимазації

- Перехід на подійну модель – замість оновлення в Update() квест змінюється лише при отриманні тригерів.
- Зменшення виділень пам’яті – зберігання квестів зроблено так, щоб не створювались їх копії.
- Рідке використання корутин – навіть тоді коли нам потрібно використовувати Update() для короткочасної праці кожен кадр, то замість цього ми використовуємо Coroutines.

Аналіз продуктивності після оптимізації

Спираючись на аналіз продуктивності в минулому пункті можна зробити висновки що:

- Середнє використання CPU: близько 1 мс (що в 2 раза менше від продуктивності без оптимізації)

Завдяки переходу на подійну систему та зменшенню зайвих обчислень система стала набагато швидшою.

Висновок до розділу 4

У цьому розділі було проведено тестування backend системи в умовах реального ігрового середовища. Перевірено коректність роботи квестів, діалогів та оновлення станів кроків. Здійснено аналіз продуктивності за допомогою Unity Profiler, що дозволило виявити найбільш ресурсоємні ділянки коду. Після впровадження оптимізації зафіксовано зменшення навантаження на процесор та покращення загальної стабільності роботи системи.

ВИСНОВКИ

Дана робота була присвячена розробці автоматизованої системи управління квестами, яка дозволяє гнучко керувати ігровими сценаріями, підвищуючи ефективність розробки інтерактивного контенту. Було проведено аналіз сучасних підходів до створення подібних систем, у результаті чого виявлено, що багато існуючих рішень або надмірно ускладнені, або не забезпечують достатньої гнучкості для розширення функціоналу.

У ході розробки було запропоновано концептуальну модель системи, що включає модульну архітектуру, де кожен компонент відповідає за окрему частину обробки квестів. Була реалізована система подій, замість постійного оновлення у кожному кадрі, що дозволило знизити навантаження на процесор і мінімізувати непотрібні обчислення. Для управління залежностями інтегровано Zenject, що значно покращило підтримку та розширюваність коду.

Окрему увагу приділено оптимізації продуктивності, що включало зменшення частоти викликів перевірок стану квестів. Тестування продуктивності проведене за допомогою Unity Profiler, показало суттєве зменшення навантаження на процесор, що позитивно позначилося на стабільності системи.

Розроблена система продемонструвала високу ефективність у керуванні квестами, забезпечуючи зручний механізм для створення, оновлення та відстеження стану завдань. Вона може бути інтегрована у широкий спектр ігрових проектів, адаптуючись під специфічні вимоги розробників.

Таким чином, реалізоване рішення дозволяє автоматизувати процес керування квестами, зменшуючи навантаження на розробників і покращуючи загальну продуктивність гри. Запропонований підхід може бути розширений для використання у складніших сценаріях, що підтверджує його перспективність для подальшого розвитку.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Розробка ігор: технології та інструменти. *ДУІКТ*. URL: https://duikt.edu.ua/ua/news-1-570-11350-rozrobka-igor-tehnologii-ta-instrumenti_kafedra-informaciynih-sistem-ta-tehnologiy (дата звернення: 09.10.2024).

2. 10 найкращих ігрових рушіїв. *Ulab - SumDU*. URL: https://ulab.sumdu.edu.ua/uk/10-najkrashhih-igrovih-rushiiv (дата звернення: 14.10.2024).

3. Benjaminov A. Implementing a Scalable Quest System. *Medium*. URL: https://medium.com/better-programming/implementing-a-scalable-quest-system-7f36ea4cfe22 (дата звернення: 03.11.2024).

4. Contributors to Wikimedia projects. Nonlinear gameplay - Wikipedia. *Wikipedia*. URL: https://en.wikipedia.org/wiki/Nonlinear_gameplay (дата звернення: 16.10.2024).

5. Gregory J. Game Engine Architecture. 3-тє вид. A K Perets/CRC Press, 2018. 1240 с.

6. Grimoire K. V. RPGs and their Dialogue Systems. *Medium*. URL: https://medium.com/@kaptainviciorious/rpgs-and-their-dialogue-systems-73307caa2b81 (дата звернення: 18.10.2024).

7. Introduction of Object Oriented Programming. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/introduction-of-object-oriented-programming/ (дата звернення: 14.11.2024).

8. McShaffry M., Graham D. Game Coding Complete. 4-те вид. Cengage Learning PTR, 2012. 960 с.

9. Microsoft. C# Guide. *C# Documentation*. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 16.11.2024).

10. Nystrom R. Game Programing Patterns. Genever Benning, 2014. 354 с.

11. Quest Systems. *Cesar Gonzalez*. URL: https://cesargonzalezgames.com/portfolio/quest-systems/#1-the-quest-system-s-structure-shapes-the-designers-ambitions (дата звернення: 05.11.2024).

12. Unity Perfomance Optimization Techniques. *Stack Overflow*. URL: https://stackoverflow.com/questions/tagged/unity3d (дата звернення: 21.01.2025).

13. Unity Profiler: Performance Analysis. *Unity Technologies*. URL: https://docs.unity3d.com/Manual/Profiler.html (дата звернення: 11.01.2025).

14. Unity User Manual. *Unity Technologies*. URL: https://docs.unity3d.com/Manual/UnityManual.html (дата звернення: 01.11.2024).

15. Unity vs Unreal: What to Choose in 2025?. *Game Art Outsourcing Studio*. URL: https://rocketbrush.com/blog/unity-vs-unreal-engine-which-one-should-you-choose-in-2024 (дата звернення: 15.10.2024).

16. Valente M. Code Performance Optimization: Effective Memory Usage & CPU Optimization,. A-List Publishing, 2003. 400 с.

17. Zenject Documentation. *Zenject Dependency Injection for Unity*. URL: https://github.com/modesttree/Zenject (дата звернення: 20.12.2024).

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет імені В. Н. Каразіна

Навчально-науковий інститут комп'ютерних наук та штучного інтелекту
Кафедра комп'ютерних систем та робототехніки
Рівень вищої освіти (освітньо-кваліфікаційний рівень) **Бакалавр**
Галузь знань: 15 – Автоматизація та приладобудування
Спеціальність: 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітня програма «Автоматизація та комп'ютерно-інтегровані технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних
систем та робототехніки
к. ф.-м. н., доц. ХРУСЛОВ М. М.
«02» жовтня 2024року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

САВЧУКА Владислава Артемовича
(прізвище, ім'я, по батькові студента)

1. Тема роботи **BACKEND-СИСТЕМА ДЛЯ УПРАВЛІННЯ СЦЕНАРІЯМИ ТА ВИБОРАМИ КОРИСТУВАЧІВ У ГРАФІЧНИХ ПРИГОДНИЦЬКИХ ІГРАХ**

керівник роботи **Бикова Тетяна Володимирівна, кандидат технічних наук, доцент**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від ***16 квітня 2025 року №4101-5/962***

2. Строк подання студентом роботи ***30 травня 2025 року***

3. Перелік питань, які потрібно розробити)

1. Проаналізувати існуючі рішення для управління сценаріями в графічних пригодницьких іграх.
2. Розробити архітектуру backend системи, яка базується на сучасних підходах до побудови програмного забезпечення.
3. Реалізувати функціонал для роботи з інтерактивними діалогами, виборами користувачів та їх наслідками.
4. Інтегрувати системи залежностей з використанням Zenject для забезпечення гнучкості та масштабованості проекту.
5. Провести тестування backend системи на базі створеної гри та оцінити її ефективність.

4. План роботи

№ з/п	Назви етапів роботи	Термін виконання етапів роботи
1	Затвердження теми роботи	05.09.2024-26.09.2024
2	Літературний огляд сучасних підходів до розробки backend систем для управління сценаріями та виборами користувачів для інтерактивних ігор	26.09.2024-18.10.2024
3	Аналіз методів управління сценаріями та виборами користувачів	19.10.2024-02.11.2024
4	Розробка концепції backend системи	03.11.2024-15.11.2024
5	Реалізація системи квестів та інтерактивних сценаріїв	16.11.2024-10.01.2025
6	Тестування backend системи, оцінка стабільності та гнучкості	11.01.2025-05.02.2025
7	Визначення обмежень системи та розробка рекомендацій щодо її вдосконалення	06.02.2025-25.02.2025
8	Підготовка та оформлення звітних матеріалів та додатків кваліфікаційної роботи. Оформлення списку літератури	26.02.2025-10.03.2025
9	Оформлення пояснювальної записки відповідно до вимог	11.03.2025-30.05.2025
10	Оформлення звіту про переддипломну практику	11.03.2025-30.05.2025

5. Дата видачі завдання ***02 жовтня 2024 року.***

Студент В. А. Савчук _____
ініціали, прізвище
підпис

Керівник роботи Т. В. Бикова _____
ініціали, прізвище
підпис

Додаток Б

Затверджую

_____

«_____»_____2025 р.

Технічне завдання
на розробку системи
«Backend-система управління сценаріями та виборами користувачів у графічних пригодницьких іграх»

1.	Введення	1.1. Назва: Backend-система управління сценаріями та виборами користувачів у графічних пригодницьких іграх. 1.2. Галузь застосування: розробка ігор.
2.	Підстава для розробки	2.1. Навчальний план за спеціальністю 151 – Автоматизація та комп'ютерно-інтегровані технології 2.2. Завдання на кваліфікаційну роботу бакалавра №4101-5/962 від 16 квітня 2025 року (представити як Додаток А до пояснювальної записки до кваліфікаційної роботи).
3.	Призначення розробки	3.1. Мета: вирішення проблеми складності та доступності реалізації інтерактивних сценаріїв у комп'ютерних іграх через впровадження модульної системи, орієнтованої на потреби інді-розробників. 3.2. Призначення розробки: надати можливість зручно створювати нелінійні квести в іграх, які забезпечують інтерактивність, при чому система повинна обладати адаптивністю, продуктивністю, та масштабованістю. 3.3. Вхідні дані розробки: налаштування квесту (послідовність кроків, налаштування кроків, локалізація тощо) 3.4. Вихідні дані розробки: працюючий квест, який може мати декілька розгалуджень сценарію.
4.	Технічні вимоги до програмного виробу	4.1. Вимоги до функціональних характеристик: інтерактивність квесту, підтримка нелінійності. 4.2. Вимоги до нефункціональних характеристик: інтуїтивно зрозумілий інтерфейс налаштування квесту. 4.3. Вимоги до надійності: забезпечувати високу продуктивність в рамках підтримки й оновлення квесту. 4.4.Вимоги до умов експлуатації: немає. 4.5. Вимоги до складу і параметрів технічних засобів: ПК.

		4.6. Вимоги до інформаційної та програмної сумісності: ігровий рушій Unity. 4.7. Вимоги до маркування та упаковки: немає. 4.8. Вимоги до транспортування і зберігання: немає. 4.9. Спеціальні вимоги: немає.
5.	Вимоги до програмної документації	Програмною документацією до виробу «Backend система для управління сценаріями та виборами користувачів у комп'ютерних іграх типу "Графічна пригодницька гра"».вважати: 1) Справжнє Технічне завдання на розробку виробу (представити у вигляді Додатку Б до пояснювальної записки до кваліфікаційної роботи). 2) Програму і методику випробувань розробленого програмного виробу (представити у вигляді Додатку В до пояснювальної записки до дипломної роботи). 3) Опис програмного виробу (представити в розділі 3 пояснювальної записки до дипломної роботи). 4) Код програми (представити в Додатку Г до пояснювальної записки до дипломної роботи).
6.	Вимоги до техніко-економічних показників	Продуктивність системи показати у вигляді графіків тестування.
7.	Стадії і етапи розробки	Затвердження теми роботи 05.09.2024-26.09.2024 Літературний огляд сучасних підходів до розробки backend систем для управління сценаріями та виборами користувачів для інтерактивних ігор 26.09.2024-18.10.2024 Аналіз методів управління сценаріями та виборами користувачів 19.10.2024-02.11.2024 Розробка концепції backend системи 03.11.2024-15.11.2024 Реалізація системи квестів та інтерактивних сценаріїв 16.11.2024-10.01.2025

		Тестування backend системи, оцінка стабільності та гнучкості	11.01.2025-05.02.2025
		Визначення обмежень системи та розробка рекомендацій щодо її вдосконалення	06.02.2025-25.02.2025
		Підготовка та оформлення звітних матеріалів та додатків кваліфікаційної роботи. Оформлення списку літератури	26.02.2025-10.03.2025
		Оформлення пояснювальної записки відповідно до вимог	11.03.2025-30.05.2025
		Оформлення звіту про переддипломну практику	11.03.2025-30.05.2025
8.	Порядок контролю і приймання програмного продукту (моделі)	1. Перевірку ходу розробки програми виконувати раз в 3 тижні. 2. Випробування програмного продукту провести відповідно до програми та методики випробувань. 3. Захист розробленої моделі провести на засіданні Атестаційної комісії. 4. Пояснювальну записку подати в електронному вигляді в 1 примірнику та в паперовому вигляді в 1 примірнику.	

Виконавець
студент групи КУ- 41
САВЧУК В. А.
_____

Замовник
д. т. н., проф.
БИКОВА Т. В.
_____

Додаток В

Програма і методика випробувань програмного виробу «Backend-система для управління сценаріями та виборами користувачів»

1. Об’єкт випробувань

1. Назва програмного виробу : «Backend система для управління сценаріями та виборами користувачів у комп'ютерних іграх типу "Графічна пригодницька гра"»

2. Галузь застосування : Розробка ігор

3. Перераховані відомості запозичуються з відповідних розділів Технічного завдання.

2. Мета випробувань

Перевірка функціональності та продуктивності програмної реалізації.

3. Загальні положення

3.1. Підстави для проведення випробувань

Підставою для проведення випробувань є наказ про призначення атестаційної комісії.

3.2. Місце і тривалість випробувань

Приймальні (приймально-здавальні) випробування проводяться на базі комп'ютерного класу кафедри в період роботи атестаційної комісії.

3.3. Обсяг випробувань

Приймальні випробування програмного виробу проводяться в обсязі відповідному цієї програми і методики випробувань.

3.4. Організації які беруть участь у випробуваннях

Приймальні випробування проводяться атестаційною комісією напередодні засідання (або в процесі засідання) за участю Замовника, Виконавця та інших осіб, присутніх на засіданні.

4. Вимоги до програми або програмного виробу

Модель повинна задовольняти наступним вимогам:

1. працювати на найбільш поширеній операційній системі Windows;

2. вимоги до надійності;

3. бути легко розширюваною;

4. елементи програми повинні бути не тісно зв’язані одні від одним для зменшення їх впливу на роботи програми під час редагування програмного коду;

5. вимоги до маркування та упаковки (не висуваються);

6. вимоги до транспортування і зберігання (не висуваються).

7. спеціальні вимоги (не висуваються).

5. Вимоги до програмної документації

Програмною документацією щодо розроблюваного програмного продукту вважати:

1. справжнє технічне завдання на розробку програми (представити як Додаток Б до пояснювальної записки до кваліфікаційної роботи);

2. Програму і методику випробувань розробленої програми (представити як Додаток В до пояснювальної записки до кваліфікаційної роботи);

3. рекомендацій щодо застосування створеної програмної стандартизації у проєктах (представити в Розділі 3 пояснювальної записки до кваліфікаційної роботи).

4. Код програми (представити як Додаток Г до пояснювальної записки до кваліфікаційної роботи).

6. Засоби і порядок випробувань

6.1 Засоби випробувань

Для проведення випробувань необхідний проект Unity, в якому буде налаштований квест з використанням нашої системи, а також сам рушій Unity.

Випробування може проводитися на технічних засобах таких як ПК або ноутбук.

6.2 Порядок проведення випробувань

Як правило, випробування проводяться в два етапи:

-ознайомчий (1-й етап);

-випробування програмного виробу (2-й етап).

Перелік перевірок, що проводяться на 1 етапі випробувань, включає в себе:

1. Перевірку комплектності програмної документації.

2. Перевірка комплектності складу програмної документації здійснюється за критерієм наявності зазначеної в ТЗ документації.

3. Перевірку комплектності складу технічних і програмних засобів.

4. Методику проведення перевірок на 1 етапі випробувань.

5. Якість програмної документації перевіряється на відповідність вимогам стандартів ЕСПД.

Перелік перевірок, що проводяться на 2 етапі випробувань, включає в себе:

1. перевірку відповідності технічних характеристик програми вимогам технічного завдання;

2. перевірку ступеня виконання функціональних вимог до програми;

3. методику проведення перевірок, що входять до переліку по 2 етапу випробувань.

1. Програма працює відповідно до умов експлуатації операційної системи MS Windows.

2. Для роботи необхідний рушій Unity.

3. Порядок проведення випробувань:

3.1. Запуск програми здійснюється за допомогою запуску сцени в рушії.

3.2. Після запуску програми необхідно слідувати вказаним рекомендаціям квесту, тобто виконувати дії, які були заздалегідь прописані як сценарій квесту.

3.3. Після виконання зазначених умов кроку квесту, стан самого квесту повинен змінитися на завершений, або крок перейде на наступний.

Для проведення випробувань пропонується тест 1, та тест 2

Тест 1

1. Перевірка правильності виконання програми;
2. Виконання умов першого кроку квесту.
3. Гра автоматично оновлює стан квесту, й переходить на наступний крок.

Рис. В.1 – результат запуску гри.

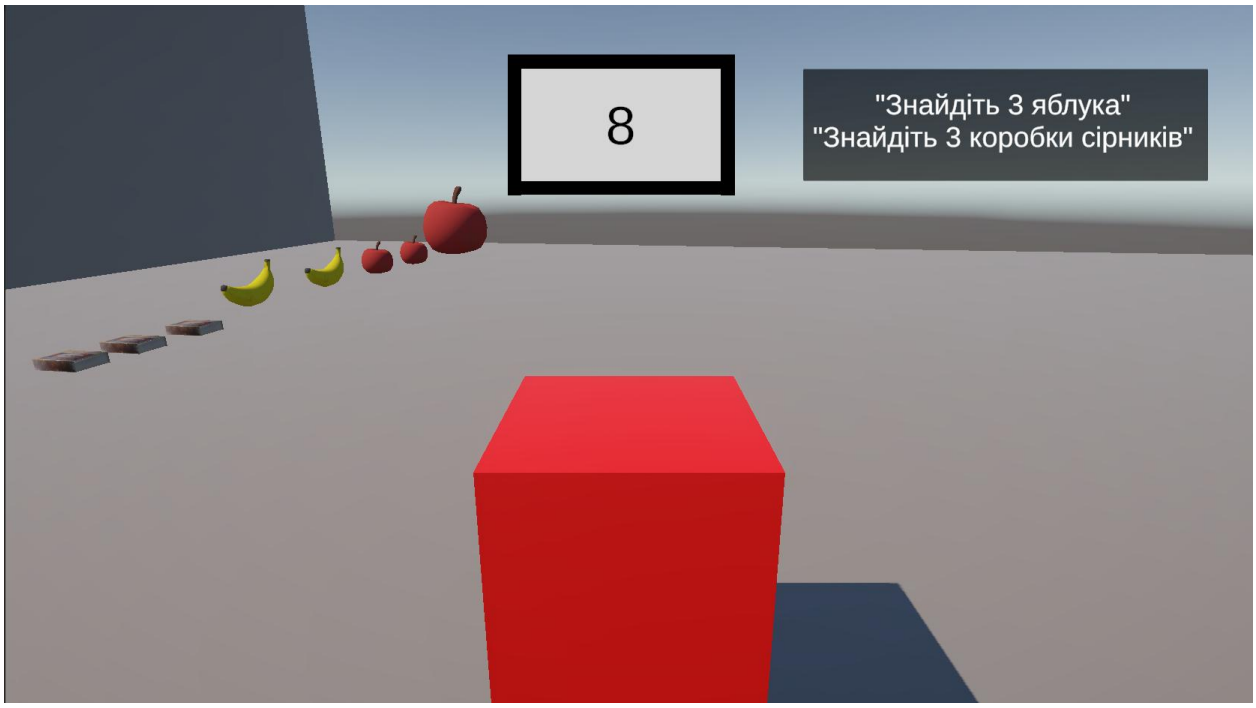

Рис. В.2 – результат оновлення й переходу кроку квесту до наступного.

Тест 2

1. Перевірка правильності виконання програми;

2. Виконання умов першого кроку квесту, але в діалозі обрати іншу відповідь.

3. Гра автоматично оновлює стан квесту, й переходить на наступний крок, але на відмінний від тесту 1.

Рис. В.3 – Обираємо інший варіант відповіді

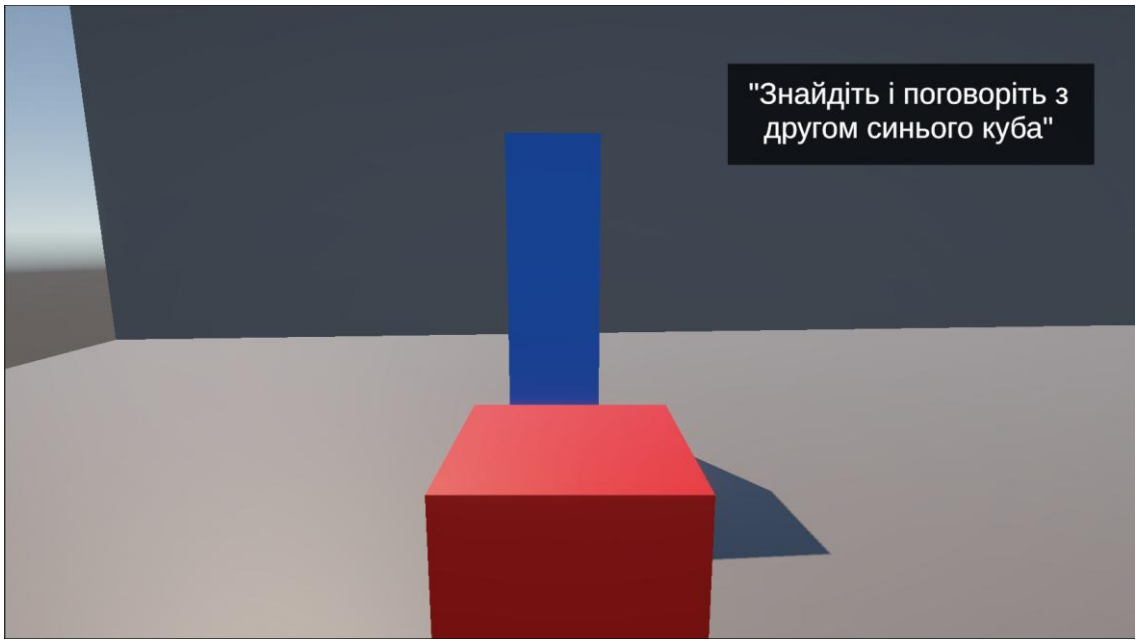

Рис. В.4 – Нелінійна зміна кроку

Отже, тест 1 пройшов успішно, бо стан кроку оновлюється як і треба, і тест 2 також можна вважати успішним, бо система оновила крок до іншої гілки сценарію. Зважаючи на те, що обидва тести пройшли перевірки успішно, нашу систему можна вважати працюючою.

Виконавець: студент групи КУ-41, Савчук Владислав _____

Додаток Г

Лістинг Г.1 - QuestManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using TMPro;
using UnityEngine;

public class QuestManager : MonoBehaviour
{
    [SerializeField] private TMP_Text _questNamePanel;
    [SerializeField] private GameObject _questDescriptionPanel;
    private List<Quest> _quests;
    private bool _isQuestRun;
    private int _currentQuestIndex;
    public event Action OnChapterPartComplete

    private void Awake()
    {
        InitializeQuests();
    }

    private void InitializeQuests()
    {
        _quests = new List<Quest>(

FindObjectsByType<MonoBehaviour>(FindObjectsSortMode.None).OfType<Quest>()
        );
        RunQuest(0);
    }

    private void RunQuest(int questIndex)
    {
        _currentQuestIndex = questIndex;
        _quests[_currentQuestIndex].OnComplete += QuestOnComplete;
        _quests[_currentQuestIndex].RunQuest();
        _isQuestRun = true;
    }

    public void QuestOnComplete()
    {
        _currentQuestIndex = -1;
        _isQuestRun = false;
        StartCoroutine(HideDescriptionPanel(3));
    }

    private IEnumerator HideDescriptionPanel(int delay) {
        yield return new WaitForSeconds(delay);
    }

    public void UpdateQuestDescription(string description)
    {
        _questDescriptionPanel.SetActive(true);
        _questDescriptionPanel.transform.GetChild(0).GetComponent<TMP_Text>().text =
description;
    }

    public Quest FindActiveQuest() {
        return _quests[_currentQuestIndex];
    }
}
```

Лістинг Г.2 - BaseQuest.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;

public abstract class BaseQuest : MonoBehaviour
{
    public abstract QuestManager QuestManager{ get; set; }
    public abstract string QuestName{ get; set; }
    public abstract string GeneralQuestDescription{ get; set; }
    public abstract List<QuestStep> Steps{ get; set; }
    public abstract int CurrentStepIndex{ get; set; }
    public abstract TextAsset QuestLocalizationFile{ get; set; }
    public abstract QuestLocalizationLoader QuestLocalizationLoader{ get; set; }
    public abstract string CurrentLanguage{ get; set; }
    public abstract event Action OnComplete;
    public abstract void RunQuest();
}
```

Лістинг Г.3 - BaseQuestStep.cs

```
public abstract class BaseQuestStep
{
       public abstract string Description{ get; set;
       public abstract int ActionId{ get; set; }
       public abstract int[] NextStepIds{ get; set; }

       public abstract void OnStart();
       public abstract void OnComplete();
}
```

Лістинг Г.4 - Quest.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;

public class Quest : BaseQuest
{
    private Dictionary<int, HashSet<int>> _completedStepsByActionId = new
Dictionary<int, HashSet<int>>();
    [SerializeField] private string _questName;
    [SerializeField] private string _generalQuestDescription;
    [SerializeField] private List<QuestStep> _steps;
    [SerializeField] private TextAsset _questLocalizationFile;
    private QuestLocalizationLoader _questLocalizationLoader;
    private QuestManager _questManager;
    private int _currentStepIndex;
    private string _currentLanguage;
    private QuestUtils _questUtils;
    [SerializeField] public Item RewardItem;

    public override string QuestName
    {
      get = > _questName;
      set = > _questName = value;
    }

        public override string GeneralQuestDescription
    {
      get = > _generalQuestDescription;
      set = > _generalQuestDescription = value;
    }

        public override List<QuestStep> Steps
```

```
{
  get = > _steps;
  set = > _steps = value;
}

    public override int CurrentStepIndex
{
  get = > _currentStepIndex;
  set = > _currentStepIndex = value;
}

    public override TextAsset QuestLocalizationFile
{
  get = > _questLocalizationFile;
  set = > _questLocalizationFile = value;
}

    public override QuestLocalizationLoader QuestLocalizationLoader
{
  get = > _questLocalizationLoader;
  set = > _questLocalizationLoader = value;
}
    public override string CurrentLanguage
{
  get = > _currentLanguage;
  set = > _currentLanguage = value;
}
    public override QuestManager QuestManager
{
  get = > _questManager;
  set = > _questManager = value;
}

public override event Action OnComplete;

public override void RunQuest()
{
    QuestLocalizationLoader = new QuestLocalizationLoader();
    QuestLocalizationLoader.LoadLocalization(_questLocalizationFile);
    QuestManager = FindAnyObjectByType<QuestManager>();
    _questUtils = FindAnyObjectByType<QuestUtils>();
    GameSettingsManager.OnLanguageChanged += UpdateLanguage;
    LoadProgress();
}

public void CompleteQuest()
{
    OnComplete ? .Invoke();
    QuestManager.QuestOnComplete();
}

public void FailedQuest()
{
    QuestManager.QuestOnComplete();
}

private void OnDestroy()
{
    GameSettingsManager.OnLanguageChanged -= UpdateLanguage;
}

private void GetLocalizationLanguage()
{
    switch (PlayerPrefs.GetInt("languageKey", 0))
    {
```

```
            case 0:
                CurrentLanguage = "en";
                break;
            case 6:
                CurrentLanguage = "ru";
                break;
            case 7:
                CurrentLanguage = "ua";
                break;
            default:
                CurrentLanguage = "en";
                break;
            }
        }

        private void UpdateLanguage()
        {
            List<QuestStep> result = new List<QuestStep>();

            foreach(var step in _steps)
            {
                if (step.ActionId == CurrentStepIndex)
                {
                    result.Add(step);
                }
            }

            GetLocalizationLanguage();
            UpdateStepDescription(result);
        }

        private void LoadProgress()
        {
            string key = $"QuestProgress_{_questName}";
            if (PlayerPrefs.HasKey(key))
            {
                string json = PlayerPrefs.GetString(key);
                var progress = JsonUtility.FromJson<QuestProgress>(json);

                CurrentStepIndex = progress.CurrentStepIndex;
                Debug.Log($"Прогресс квеста \"{_questName}\" завантажений. Поточний шаг:
{CurrentStepIndex}");

                UpdateStep();
            }
            else
            {
                Debug.Log($"Прогресс для квеста \"{_questName}\" не знайдений. Квест
починається з початку");
                CurrentStepIndex = 0;
                UpdateStep();
            }
        }

        public void SaveProgress()
        {
            var progress = new QuestProgress
            {
              QuestName = _questName,
              CurrentStepIndex = _currentStepIndex
            };

            string json = JsonUtility.ToJson(progress);
            PlayerPrefs.SetString($"QuestProgress_{_questName}", json);
            PlayerPrefs.Save();
```

```
        Debug.Log($"Прогресс квеста \"{_questName}\" збережений: {json}");
    }

    private void UpdateStepDescription(List<QuestStep> currentSteps)
    {
        string localizedDescription = "";
        foreach(var currentStep in currentSteps) {
            localizedDescription +=
QuestLocalizationLoader.GetStepDescription(currentStep.ActionId, CurrentLanguage) +
"\n";
            break;
        }
        QuestManager.UpdateQuestDescription(localizedDescription);
    }

    private void UpdateStep()
    {
        var currentSteps = GetStepByActionId(_steps, CurrentStepIndex);
        GetLocalizationLanguage();
        UpdateStepDescription(currentSteps);
        foreach(var currentStep in currentSteps)
        {
            currentStep.OnStart();
            if (currentStep.IsTimer)
            {
                _questUtils.RunQuestTimer(currentStep.TimerDuring);
            }
        }
    }

    public List<QuestStep> GetStepByActionId(List<QuestStep> steps, int actionId)
    {
        List<QuestStep> result = new List<QuestStep>();

        foreach(var step in steps)
        {
            if (step.ActionId == actionId)
            {
                result.Add(step);
            }
        }

        return result;
    }

    private bool CanProceedToNextStep(List<QuestStep> steps)
    {
        bool hasRequiredSteps = false;
        bool hasCompletedRequiredStep = true;
        bool hasAnyCompletedStep = false;

        foreach(var step in steps)
        {
            if (step.IsRequired)
            {
                hasRequiredSteps = true;
                if (!step.IsComplete)
                {
                    hasCompletedRequiredStep = false;
                }
            }

            if (step.IsComplete)
            {
```

```
                hasAnyCompletedStep = true;
            }
        }

        if (hasRequiredSteps)
        {
            return hasCompletedRequiredStep;
        }

        return hasAnyCompletedStep;
    }

    public void CompleteStep(int actionId)
    {
        if (!_completedStepsByActionId.ContainsKey(actionId))
        {
            _completedStepsByActionId[actionId] = new HashSet<int>();
        }

        List<QuestStep> result = new List<QuestStep>();

        foreach(var step in _steps)
        {
            if (step.ActionId == CurrentStepIndex)
            {
                result.Add(step);
            }
        }

        if (CanProceedToNextStep(result)) {
            foreach(var step in result)
            {
                Debug.Log($"Завершення шагу '{step.Description}'");
                step.OnComplete();
                if (step.IsTimer)
                {
                    _questUtils.StopQuestTimer();
                }
            }
            CurrentStepIndex = actionId;

            if (CurrentStepIndex < _steps.Count)
            {
                UpdateStep();
                Debug.Log("Оновлення опису квеста");
                SaveProgress();
            }
            else
            {
                Debug.Log("Квест завершений");
                QuestManager.QuestOnComplete();
            }
        }
        else {
            return;
        }
    }

#if UNITY_EDITOR
    [ContextMenu("Reset Progress")]
#endif
    public void ResetProgress()
    {
        PlayerPrefs.DeleteKey($"QuestProgress_{_questName}");
        Debug.Log($"Прогрес для квеста \"{_questName}\" зброшений.");
```

```
    }

#if UNITY_EDITOR
    [ContextMenu("Validate Steps")]
#endif
    public void ValidateSteps()
    {
        var actionIds = new HashSet<int>();
        foreach(var step in _steps)
        {
            if (actionIds.Contains(step.ActionId))
            {
                Debug.LogError($"Повторюющийся ActionId: {step.ActionId} в кроці
\"{step.Description}\".");
            }
            else
            {
                actionIds.Add(step.ActionId);
            }
        }

        Debug.Log("Перевірка кроків завершена.");
    }

#if UNITY_EDITOR
    [ContextMenu("Sort Steps by ActionId")]
#endif
    public void SortSteps()
    {
        _steps.Sort((a, b) = > a.ActionId.CompareTo(b.ActionId));
        Debug.Log("Кроки відсортовані по ActionId.");
    }
}
```

Лістинг Г.5 - QuestStep.cs

```
using System;
using UnityEngine;
using UnityEngine.Events;

[Serializable]
public class QuestStep : BaseQuestStep
{
    [Header("Опис шагу")]
    [SerializeField] private string _description;

    [Header("Ідентифікатор шагу")]
    [SerializeField] private int _actionId;

    [Header("Наступні шаги")]
    [SerializeField] private int[] _nextStepIds;

    [Header("Чи є таймер на виконання шагу")]
    [SerializeField] public bool IsTimer = false;

    [Header("Якщо є то скільки часу таймер")]
    [SerializeField] public float TimerDuring;

    [Header("Подія початку шагу")]
    [SerializeField] private QuestStepEvent _onStart;

    [Header("Подія завершення шагу")]
    [SerializeField] private UnityEvent _onComplete;

    [Header("Чи є шаг обов'язковим")]
    [SerializeField] private bool _isRequired;
```

```
    [Header("Чи є шаг виповненим")]
    [SerializeField] private bool _isComplete = false;

    public override string Description
    {
      get = > _description;
      set = > _description = value;
    }

        public override int ActionId
    {
      get = > _actionId;
      set = > _actionId = value;
    }

        public override int[] NextStepIds
    {
      get = > _nextStepIds;
      set = > _nextStepIds = value;
    }

        public bool IsRequired{
          get = > _isRequired;
          set = > _isRequired = value;
    }

        public bool IsComplete
    {
      get = > _isComplete;
      set = > _isComplete = value;
    }

        public override void OnStart()
    {
        Debug.Log($"Початок шагу: {_description}");
        _onStart ? .Invoke(this);
    }

    public override void OnComplete()
    {
        Debug.Log($"Кінець шагу: {_description}");
        _onComplete ? .Invoke();
    }

    public void SetComplete(bool isComplete)
    {
        _isComplete = isComplete;
    }
}

[System.Serializable]
public class QuestStepEvent : UnityEvent<QuestStep> {}
```

Лістинг Г.6 - QuestTimer.cs

```
using System.Collections;
using TMPro;
using UnityEngine;

public class QuestTimer : MonoBehaviour
{
    [SerializeField] private QuestManager _questManager;
    private Coroutine _activeCoroutine;
```

```
    public void Run(float duringTimer)
    {
        gameObject.SetActive(true);
        if (_activeCoroutine != null)
        {
            StopCoroutine(_activeCoroutine);
        }
        _activeCoroutine = StartCoroutine(Timer(duringTimer));
    }

    private IEnumerator Timer(float duringTimer)
    {
        TMP_Text timer = gameObject.transform.GetChild(1).GetComponent<TMP_Text>();
        timer.text = duringTimer.ToString();
        float time = duringTimer;

        for (int i = 0; i < duringTimer; i++)
        {
            yield return new WaitForSeconds(1);
            time--;
            timer.text = time.ToString();
        }

        var quest = _questManager.FindActiveQuest();
        quest.CompleteStep();
    }

    public void StopTimer()
    {
        if (_activeCoroutine != null)
        {
            StopCoroutine(_activeCoroutine);
            _activeCoroutine = null;
        }

        gameObject.SetActive(false);
    }
}
```

Лістинг Г.7 - QuestProgress.cs

```
using System;

[Serializable]
public class QuestProgress
{
      public string QuestName;
      public int CurrentStepIndex;
}
```

Лістинг Г.8 - QuestUtils.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;

public class QuestUtils : MonoBehaviour
{
    private InventoryInteractor _inventory;
    [SerializeField] public QuestTimer _questTimer;

    private void Start()
    {
```

```
        _inventory = FindAnyObjectByType<InventoryManager>().Interactor;
    }

    public void CheckItemOnInventory(int idItem, int countItem, Action onFound)
    {
        List<ItemData> items = _inventory.GetAllItems();
        foreach(ItemData item in items)
        {
            if (item.Id == idItem && item.CountItem == countItem)
            {
                onFound ? .Invoke();
            }
        }
    }

    public void RemoveItemFromInventory(int idItem, int countItem)
    {
        _inventory.RemoveItem(idItem, countItem);
    }

    public void GiveReward(Item item)
    {
        _inventory.AddItem(item);
    }

    public void RunQuestTimer(float duringTimer) {
        _questTimer.Run(duringTimer);
    }

    public void StopQuestTimer() {
        _questTimer.StopTimer();
    }
}
```

Лістинг Г.9 - ParseDialogFile.cs

```
using System.Xml;
using UnityEngine;
using Zenject;

public class ParseDialogFile
{
    private int _languageIndex;
    private int _soundLanguageIndex;
    private GameSettingsInteractor _gameSettingsInteractor;

    [Inject]
    public ParseDialogFile(GameSettingsInteractor gameSettingsInteractor)
    {
        _languageIndex = gameSettingsInteractor.GetLanguageIndex();
        _soundLanguageIndex = gameSettingsInteractor.GetSoundLanguageIndex();
        _gameSettingsInteractor = gameSettingsInteractor;
    }

    public DialogNode GetDialogTree(TextAsset textAsset)
    {
        XmlDocument xmlDoc = new XmlDocument();
        xmlDoc.LoadXml(textAsset.text);
        XmlNode rootDialogNode = xmlDoc.SelectSingleNode("dialog");
        return CreateDialogTree(rootDialogNode);
    }

    private DialogNode CreateDialogTree(XmlNode dialogNode, DialogNode parent =
null)
    {
```

```
        _languageIndex = _gameSettingsInteractor.GetLanguageIndex();
        _soundLanguageIndex = _gameSettingsInteractor.GetSoundLanguageIndex();
        var node = new DialogNode
        {
            Parent = parent,
            Name =
dialogNode.SelectSingleNode($"name/{GetLanguageTag(_languageIndex)}") ? .InnerText ?
? "Unknown:",
            Message =
dialogNode.SelectSingleNode($"message/{GetLanguageTag(_languageIndex)}") ?
.InnerText ? ? "No message available.",
            ShortName =
dialogNode.SelectSingleNode($"short_name/{GetLanguageTag(_languageIndex)}") ?
.InnerText
        };

        XmlNode actionNode = dialogNode.SelectSingleNode("action");
        if (actionNode != null && int.TryParse(actionNode.Attributes["id"] ? .Value,
out int actionId))
        {
            node.ActionId = actionId;
        }

        XmlNode voiceNode =
dialogNode.SelectSingleNode($"voice/{GetSoundLanguageTag(_soundLanguageIndex)}");
        if (voiceNode != null)
        {
            string audioPath = $"TalkSounds/{voiceNode.InnerText}";
            node.VoiceClip = Resources.Load<AudioClip>(audioPath);
        }

        foreach(XmlNode childNode in dialogNode.SelectNodes("dialog"))
        {
            var child = CreateDialogTree(childNode, node);
            node.Children.Add(child);
        }

        return node;
    }

    private string GetLanguageTag(int languageIndex)
    {
        return languageIndex switch
        {
            0 = > "uk", // English
                6 = > "ru", // Russian
                7 = > "ua", // Ukrainian
                _ = > "uk", // Default to English
        };
    }

    private string GetSoundLanguageTag(int soundLanguageIndex)
    {
        return soundLanguageIndex switch
        {
            0 = > "uk", // English
                1 = > "ru", // Russian
                2 = > "ua",// Ukrainian
                _ = > "uk", // Default to English
        };
    }
}
```

Лістинг Г.10 - DialogNode.cs

```
using System.Collections.Generic;
using UnityEngine;

public class DialogNode
{
    public string ShortName{ get; set; }
    public string Name{ get; set; }
    public string Message{ get; set; }
    public List<DialogNode> Children{ get; set; }
    public int ? ActionId{ get; set; }
    public AudioClip VoiceClip{ get; set; }
    public DialogNode Parent{ get; set; }

        public DialogNode()
    {
        ActionId = -1;
        Children = new List<DialogNode>();
    }
}
```

Лістинг Г.11 - DialogPresenter.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;
using Zenject;

public enum DialoguePresenterState
{
    Await,
    Talk,
    Choice,
    Finish
}

public class DialogPresenter : MonoBehaviour
{
    [Header("Components")]
    [SerializeField] private DialogView _dialogueView;
    [SerializeField] public TextAsset[] _dialoguesArr;
    [SerializeField] private MovementPlayer _movementPlayer;

    private uint _currentDialogue;
    private DialogNode _currentNode;
    public DialoguePresenterState DialogueState = DialoguePresenterState.Await;
    private bool _isMessagePrinting;

    private GameSettingsInteractor _gameSettingsInteractor;
    private ParseDialogFile _parseDialogFile;

    public event Action OnDialogueStart;
    public event Action<int ? > OnDialogEnd;

    [Inject]
    public void Construct(GameSettingsInteractor gameSettingsInteractor,
ParseDialogFile parseDialogFile)
    {
        _gameSettingsInteractor = gameSettingsInteractor;
        _parseDialogFile = parseDialogFile;
    }

    private void OnEnable()
    {
        GameSettingsManager.OnLanguageChanged += ChangedLanguage;
        GameSettingsManager.OnSoundLanguageChanged += ChangedLanguage;
```

```
        _dialogueView.OnFinishMessage += StopPrinting;
    }

    private void OnDisable()
    {
        GameSettingsManager.OnLanguageChanged -= ChangedLanguage;
        GameSettingsManager.OnSoundLanguageChanged -= ChangedLanguage;
        _dialogueView.OnFinishMessage -= StopPrinting;
    }

    private void Update()
    {
        if (PauseMenu.IsGamePaused || DialogueState == DialoguePresenterState.Await)
            return;

        if (Input.GetMouseButtonDown(0))
        {
            NextMessage();
        }
    }

    private void ChangedLanguage() {
        var path = GetCurrentNodePath();

        _currentNode =
_parseDialogFile.GetDialogTree(_dialoguesArr[_currentDialogue]);

        _currentNode = TraverseNodePath(_currentNode, path);
    }

    private List<int> GetCurrentNodePath()
    {
        var path = new List<int>();
        var current = _currentNode;

        while (current != null && current.Parent != null)
        {
            int index = current.Parent.Children.IndexOf(current);

            path.Insert(0, index);
            current = current.Parent;
        }

        return path;
    }

    private DialogNode TraverseNodePath(DialogNode root, List<int> path)
    {
        var current = root;

        foreach(var index in path)
        {
            current = current.Children[index];
        }

        return current;
    }

    public void StartDialogue()
    {
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
        _movementPlayer.enabled = false;

        if (DialogueState == DialoguePresenterState.Finish)
```

```
        {
            DialogueState = DialoguePresenterState.Talk;
        }

        _currentNode =
_parseDialogFile.GetDialogTree(_dialoguesArr[_currentDialogue]);

        _dialogueView.SetPresenter(this);
        OnDialogueStart ? .Invoke();
        _dialogueView.StartDialogue(_currentNode.Message, _currentNode.Name,
_currentNode.VoiceClip);
        DialogueState = DialoguePresenterState.Talk;
        _isMessagePrinting = true;
    }

    public void NextMessage()
    {
        if (DialogueState != DialoguePresenterState.Talk || DialogueState ==
DialoguePresenterState.Choice)
            return;

        if (_isMessagePrinting)
        {
            _dialogueView.InterruptAnimation();
            return;
        }

        if (_currentNode.Children.Count == 0)
        {
            FinishDialogue();
            return;
        }

        GoToChildMessage();
    }

    private void GoToChildMessage()
    {
        if (_currentNode.Children.Count == 1)
        {
            _currentNode = _currentNode.Children[0];
            _dialogueView.NextMessage(_currentNode.Message, _currentNode.Name,
_currentNode.VoiceClip);
            _isMessagePrinting = true;
            return;
        }

        List<string> shortNames = new List<string>();
        foreach(var child in _currentNode.Children)
        {
            shortNames.Add(child.ShortName);
        }
        _dialogueView.ActivateButtons(shortNames);
        DialogueState = DialoguePresenterState.Choice;
    }

    private void FinishDialogue()
    {
        _dialogueView.StopDialogue();
        DialogueState = DialoguePresenterState.Await;

        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
        _movementPlayer.enabled = true;
```

```
        _currentDialogue++;
        if (_currentDialogue >= _dialoguesArr.Length)
        {
            DialogueState = DialoguePresenterState.Finish;
            _currentDialogue = 0;
        }
        if (_currentNode.ActionId > 0) {
            OnDialogEnd.Invoke(_currentNode.ActionId);
        }
    }

    public void SwitchBranch(int index)
    {
        _currentNode = _currentNode.Children[index];
        DialogueState = DialoguePresenterState.Talk;
        _dialogueView.NextMessage(_currentNode.Message, _currentNode.Name,
_currentNode.VoiceClip);
        _isMessagePrinting = true;
    }

    private void StopPrinting()
    {
        _isMessagePrinting = false;
    }
}
```

Лістинг Г.12 - QuestLocalizationLoader.cs

```
using UnityEngine;
using System.IO;
using System.Linq;

public class QuestLocalizationLoader
{
    private QuestLocalization _questLocalization;

    public void LoadLocalization(TextAsset localizationFile)
    {
        string extension = Path.GetExtension(localizationFile.name);
        Debug.Log("Extension: " + extension);
        var parser = LocalizationParserFactory.GetParser(extension);
        _questLocalization = parser.Parse(localizationFile.text);
    }

    public string GetStepDescription(int stepId, string language)
    {
        var steps = _questLocalization.Steps.Where(s = > s.Id == stepId).ToList();

        var descriptions = steps
            .Where(s = > s.Description.ContainsKey(language))
            .Select(s = > s.Description[language])
            .ToList();

        return descriptions.Any() ? string.Join("\n", descriptions) : "Description
not found";
    }

    public string GetQuestName(string language)
    {
        return _questLocalization.QuestName.ContainsKey(language)
            ? _questLocalization.QuestName[language]
            : "Quest Name not found";
    }
}
```

Лістинг Г.13 - QuestStepLocalization.cs

```
using System.Collections.Generic;
using System.Runtime.Serialization;

[System.Serializable]
[DataContract(Namespace = "")]
public class QuestStepLocalization
{
	[DataMember]
	public int Id{ get; set; }
		[DataMember]
	public Dictionary<string, string> Description{ get; set; } = new
Dictionary<string, string>();
}
```

Лістинг Г.14 - QuestLocalization.cs

```
using System.Collections.Generic;
using System.Runtime.Serialization;

[System.Serializable]
[DataContract(Namespace = "")]
public class QuestLocalization
{
	[DataMember]
	public Dictionary<string, string> QuestName{ get; set; } = new
Dictionary<string, string>();
	[DataMember]
	public List<QuestStepLocalization> Steps{ get; set; } = new
List<QuestStepLocalization>();
}
```

Лістинг Г.15 - LocalizationParserFactory.cs

```
using System;
using System.Collections.Generic;

public class LocalizationParserFactory
{
    private static Dictionary<string, ILocalizationParser> parsers = new
Dictionary<string, ILocalizationParser>
    {
        { ".json", new JsonLocalizationParser() },
        { ".csv", new CsvLocalizationParser() }
    };

    public static ILocalizationParser GetParser(string fileExtension)
    {
        if (parsers.TryGetValue(fileExtension.ToLower(), out var parser))
        {
            return parser;
        }
        throw new NotSupportedException($"Unsupported localization format:
{fileExtension}");
    }
}
```

Лістинг Г.16 - CsvLocalizationParser.cs

```
using System.IO;

public class CsvLocalizationParser : ILocalizationParser
{
    public QuestLocalization Parse(string content)
    {
        QuestLocalization localization = new QuestLocalization();
        using (StringReader reader = new StringReader(content))
```

```
        {
            string line;
            while ((line = reader.ReadLine()) != null)
            {
                var parts = line.Split(',');
                if (parts.Length < 3) continue;

                int id;
                if (!int.TryParse(parts[0], out id)) continue;

                var step = new QuestStepLocalization{ Id = id };
                step.Description["en"] = parts[1];
                step.Description["ru"] = parts[2];
                if (parts.Length > 3) step.Description["ua"] = parts[3];

                localization.Steps.Add(step);
            }
        }
        return localization;
    }
}
```

Лістинг Г.17 - JsonLocalizationParser.cs

```
using Newtonsoft.Json;

public class JsonLocalizationParser : ILocalizationParser
{
    public QuestLocalization Parse(string content)
    {
        return JsonConvert.DeserializeObject<QuestLocalization>(content);
    }
}
```

Лістинг Г.18 - ILocalizationParser.cs

```
public interface ILocalizationParser
{
  QuestLocalization Parse(string content);
}
```